University of Zurich[UZH]

Department of Informatics

Automatic Transfer Function Generation and Extinction-Based Approaches in Direct Volume Visualization

A dissertation submitted to the Faculty of Economics, Business Administration and Information Technology of the University of Zurich

for the degree of
Doctor of Science (Ph.D.)

by
Philipp Christoph Schlegel
from Sevelen SG and Winterthur ZH

Accepted on the recommendation of

Prof. Dr. R. Pajarola
Prof. Dr. R. Peikert

2011

The Faculty of Economics, Business Administration and Information Technology of the University of Zurich herewith permits the publication of the aforementioned dissertation without expressing any opinion on the views contained therein.

Zurich, April 4, 2012

The head of the Ph.D. program in informatics: Prof. Abraham Bernstein, Ph.D.

ABSTRACT

Direct volume visualization has become an important tool in many domains for visualizing and examining volumetric datasets. The tremendous increase in computing power of the hardware over the past years makes it possible to immediately visualize volumetric datasets obtained from scanning devices at fully interactive frame rates. However, despite this change of paradigm compared to the slow offline methods of the past, direct volume visualization suffers from disadvantages constricting an immediate, reliable analysis of volumetric datasets.

This thesis begins with an overview of different methods for direct volume visualization followed by an in-depth review of the theoretical foundation including inherent challenges. Subsequently selected state-of-the-art techniques used in this thesis are explained in detail. One challenge that all techniques have in common is the dependency on good transfer functions. Only good transfer functions allow for the right insight into the dataset permitting a reliable analysis. These transfer functions are often constructed manually in a time consuming and cumbersome trial-and-error process. We propose an automated general purpose approach for generating a set of best transfer functions based on information theory. Our algorithm appraises the information content of the images generated by a particular transfer function when rotating the dataset, as it is the case in interactive sessions. Quantifying the quality of a transfer function in this way enables a directed search for the set of best transfer functions in a feedback loop employing a combination of two different optimization algorithms. This set of best, distinct transfer functions helps the user to gain an immediate overview of each facet of a dataset.

When visualizing volumetric datasets, it is of major importance that domain experts are able to recognize small features, to distinguish the relationship and connectivity between them and to get the right perception. For this the applied illumination and shading model plays an important part. Sophisticated models including realistic looking directional shadows, ambient occlusion and color bleeding effects can greatly enhance the perception. Unfortunately common models exhibiting these effects are expensive to compute and not suitable for interactive applications. We present a method showing how these effects can be applied to GPU volume ray-casting while fully maintaining interactivity based on the original, exponential extinction coefficient of the volume rendering integral. Exploiting the fact that the original, exponential extinction coefficient is summable, our framework is built on top of a 3D summed area table that allows for quick lookups of extinction queries.

Technically volumetric datasets consist of discrete scalar or sometimes vector data. As the resolution of this data hardly ever fits the resolution of the output device, the data needs to be interpolated or reconstructed. Volume visualization methods based on 3D textures can profit from fast built-in trilinear interpolation of the hardware. However, trilinear interpolation is not the first choice when it comes to image quality. Volume splatting on the other hand is a volume visualization technique that makes it easy to integrate arbitrary interpolation schemes. The performance of volume splatting is directly related to the applied interpolation scheme and the resulting interpolation kernel respectively. In this thesis we introduce an algorithm for volume splatting that greatly enhances the performance by reducing the required amount of splatting operations from interpolation kernel slices. Further, we show how the image quality of volume visualization can be enhanced by using the original, exponential extinction coefficient of the volume rendering integral instead of common α-blending simplifications.

KURZFASSUNG

Die direkte Volumenvisualisierung hat sich in vielen Bereichen zu einem wichtigen Instrument entwickelt, um volumetrische Datensätze zu analysieren. Die enorme Steigerung der Leistungsfähigkeit der Hardware über die letzten Jahre ermöglicht es, solche Datensätze, wie sie z.B. von Scannern stammen, interaktiv zu visualisieren. Trotz dieses Paradigmenwechsels, verglichen mit den langsamen offline Methoden der Vergangenheit, besitzt die direkte Volumenvisualisierung Nachteile, die eine unmittelbare, zuverlässige Analyse von volumetrischen Datensätzen erschweren.

Diese Dissertation beginnt mit einem Überblick über die verschiedenen Methoden zur direkten Volumenvisualisierung, gefolgt von einer ausführlichen Rekapitulation der theoretischen Grundlagen und deren inhärenten Herausforderungen. Anschliessend werden ausgewählte Techniken, die in dieser Dissertation Verwendung finden, im Detail erklärt. Eine Schwierigkeit, die für alle Techniken gilt, ist die Abhängigkeit von guten Transferfunktionen. Nur eine gute Transferfunktion erlaubt den richtigen Einblick in den Datensatz und somit eine zuverlässige Analyse. Transferfunktionen werden häufig manuell in einem zeitaufwändigen und umständlichen Prozess durch Ausprobieren konstruiert. Wir schlagen basierend auf der Informationstheorie einen universellen, automatischen Ansatz zur Generierung einer Menge bester Transferfunktionen vor. Unser Algorithmus bewertet den Informationsgehalt der Bilder, die mit Hilfe einer bestimmten Transferfunktion generiert wurden, wenn der Datensatz rotiert wird, so wie es z.B. während einer interaktiven Session der Fall ist. Diese Bewertung ermöglicht eine gerichtete

Suche nach der Menge bester Transferfunktionen in einer Resonanzschleife unter Verwendung zweier kombinierter Optimierungsalgorithmen. Die Menge bester, unterschiedlicher Transferfunktionen gestattet dem Benutzer einen unmittelbaren Einblick in jede Facette des Datensatzes.

Bei der Visualisierung von volumetrischen Datensätzen ist es sehr wichtig, dass Experten in der Lage sind, kleine Details und deren Zusammenhang zu erkennen, sowie dass sie die richtige Wahrnehmung der Struktur des Datensatzes vermittelt bekommen. Dabei spielt das verwendete Beleuchtungsmodell eine wichtige Rolle. Ausgeklügelte Modelle, die gerichtete Schatten, Umgebungsverdeckung und Farbverlauf-Effekte beinhalten, können die Wahrnehmung stark verbessern. Leider sind solche Modelle aufwändig zu berechnen und daher für interaktive Applikationen nicht geeignet. Basierend auf dem originalen, exponentiellen Extinktionskoeffizienten des Volume Rendering Integrals präsentieren wir eine Methode, die zeigt, wie solche Effekte in GPU Volume Ray-Casting integriert werden können, ohne dabei an Interaktivität einzubüssen. Dabei wird die Tatsache ausgenützt, dass der originale, exponentielle Extinktionskoeffizient aufsummiert werden kann. Folglich basiert unser Framework auf einer 3D Summed Area Table, die schnelle Abfragen von aggregierten Extinktionskoeffizienten ermöglicht.

Technisch gesehen bestehen volumetrische Datensätze aus diskreten Skalar- oder manchmal auch aus diskreten Vektordaten. Weil die Auflösung dieser Daten kaum je der Auflösung des Ausgabegerätes entspricht, müssen die Daten interpoliert bzw. rekonstruiert werden. Visualisierungsmethoden, die auf 3D Texturen basieren, können von der schnellen, eingebauten trilinearen Interpolation der Hardware profitieren. Jedoch ist die trilineare Interpolation in Bezug auf die Bildqualität nicht die erste Wahl. Volume Splatting ist eine Visualisierungstechnik, bei der es auf einfache Weise möglich ist, beliebige Interpolationsschemata zu implementieren. Die Geschwindigkeit von Volume Splatting hängt direkt vom angewendeten Interpolationsschema und dem daraus resultierenden Interpolationskernel ab. In dieser Dissertation führen wir einen Volume Splatting Algorithmus ein, der die Geschwindigkeit durch Reduktion der notwendigen Splatting-Operationen von Kernel-Slices stark erhöhen kann. Des Weiteren zeigen wir, wie die Bildqualität von Volumenvisualisierungsmethoden durch Verwendung des originalen, exponentiellen Extinktionskoeffizienten des Volume Rendering Integrals anstelle üblicher α-Blending Vereinfachungen gesteigert werden kann.

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor Renato Pajarola for the great time I enjoyed at the Visualization and Multimedia Lab during my Ph.D. studies and for his superior support. Especially, I would like to thank him for giving me the opportunity to conduct my Ph.D. studies part time while working in the private sector and gaining experience in corporate level IT at the same time. For me this was the perfect way to get the best from both worlds.

I also would like to thank Ronny Peikert for being co-examiner of this thesis. He introduced me into the topic of volume visualization when I was attending his lecture as a computer science student at ETH.

Exceptional thanks go to Christina Kaiser and my dad for the countless hours they spent proofreading this thesis and to my family who always supported me in my professional endeavors.

Finally, I would like to thank my colleagues at the Visualization and Multimedia Lab for their contributions in all professional matters, and for the good times we had in many leisure activities.

CONTENTS

NOTATIONS

Acronyms

AO	Ambient Occlusion
BRDF	Bidirectional Reflection Distribution Function
CPU	Central Processing Unit
CS	Computer Science
CT	Computed Tomography
DVR	Direct Volume Rendering
FBO	Frame Buffer Object
FPS	Frames per Second
GLSL	OpenGL Shading Language
GPU	Graphics Processing Unit
GUI	Graphical User Interface
IVS	A volume visualization system of the Visualization and Multimedia Lab of the University of Zurich
LOD	Level-of-Detail
MRI	Magnetic Resonance Imaging
PBO	Pixel Buffer Object
PET	Positron Emission Tomography
PIXEL	Picture Element
RGB	Red Green Blue
RGBA	Red Green Blue Alpha

SAT	Summed Area Table
TF	Transfer Function
VBO	Vertex Buffer Object
VDE	Visibility-Difference Entropy
VOXEL	Volume Element

Operators and Functions

$\binom{n}{r}$	Binomial coefficient: $\frac{n!}{r!(n-r)!}$
δ	Dirac delta function: $\int_{-\infty}^{\infty} \delta(x,y,z)dxdydz = 1$ and $\delta(x,y,z) = 0$ for $x \neq 0 \vee y \neq 0 \vee z \neq 0$
∇	Gradient operator: $\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right)$
O	Big O notation: upper bound of the asymptotic growth rate
$\|S\|$	Number of elements in set S
$\|x\|$	Absolute value of number x: $\|x\| = \sqrt{x^2}$
$\|\|\mathbf{x}\|\|$	Magnitude of vector $\mathbf{x}$: $\|\|\mathbf{x}\|\| = \sqrt{\mathbf{x} \cdot \mathbf{x}}$
sinc	Sinus cardinalis: $\mathrm{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$

Common Variables

α	Alpha value
ω	Frequency
τ	Extinction coefficient
υ	Angle
FP	Footprint
M	Visibility-difference entropy metric
P	Probability
Sh	Shell
V	Volumetric dataset
W	SAT query

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Volume Visualization

Volume visualization is an umbrella term for a set of techniques developed for visualizing volumetric datasets. Volumetric datasets are acquired in many different applications such as CT, MRI, PET and ultrasound scanning, numerical simulations as well as visual arts. Typically, they consist of sampled scalar or vector data, often defined on a regular grid. Compared to triangle meshes commonly used in computer graphics, volumetric datasets do not explicitly define a surface, but disclose the entire 3D structure of the dataset. For example, a volumetric dataset of a human head (see Figure 1.1) does not only contain the outer surface of the head with the skin and the hair, but also the soft tissue, the brain, the skull and the teeth. In particular, if the dataset of the human head was acquired by a CT scanner, each position of the dataset contains the radiodensity at that position. Unfortunately, there is a priori no straightforward way to visualize such a dataset. Yet, the desire for exploring the dataset, for visualizing different aspects of the dataset or just for generating good looking images is obvious. A medical doctor looking for a dangerous aneurysm is eager to explore blood vessels inside the head or is just interested in the skull while he might be searching for a splinter. Applying an appropriate visualization technique allows for accentuating properties of the dataset making its analysis easier. Parts that in reality occur opaque can be made transparent, revealing details that would be kept hidden otherwise. Certain parts can be classified using a special color, setting them apart from the surroundings.

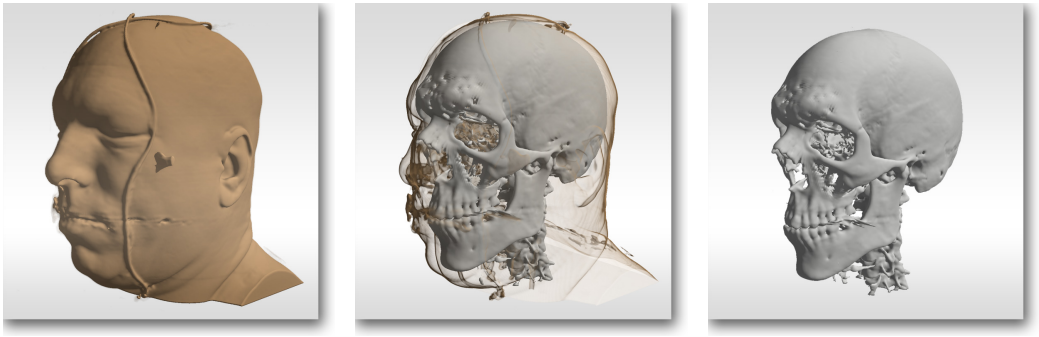


Figure 1.1: *The Visible Human volumetric dataset rendered using volume visualization with different parameters. Dataset courtesy of the National Library of Medicine, National Institutes of Health, USA.*

Volume visualization techniques are important tools for the analysis of volumetric datasets. If volume visualization was used purely for generating nice images (i.e. for feature films), it might be tolerable to wait hours until a single image is generated but if volume visualization serves the purpose of data analysis, interactivity plays an important part. Even though early approaches for volume visualization go back as far as 1972 [Wright, 1972], and early interactive approaches have reached a milestone in 1989 [Westover, 1989], it is only the last decade that showed tremendous progress in the area. Emerging programmable graphics hardware (GPUs) [Lindholm et al., 2001] and the ever increasing processing power have enabled the development of highly interactive volume visualization techniques exhibiting a good image quality at the same time. It allows domain experts such as medical doctors for immediately displaying and exploring datasets fully interactively in a quality empowering them to make the right decisions.

Subsequently an overview of the most popular volume visualization techniques is presented:

- **Volume Ray-Casting**

  The basic idea of volume ray-casting (Figure 1.2(a)) is to shoot rays from the eye of the viewer towards the volumetric dataset and tracing these rays. In particular, there is an image plane between the eye of the viewer and the dataset where the final image will be synthesized. This image plane is subdivided into discrete units called pixels. In order to synthesize the final image, the color for each of these pixels needs to be computed. Consequently, exactly one ray per pixel is shot from the eye of the viewer through the image plane into the object space of the volumetric dataset. To obtain the final color for a pixel, the respective ray is traced on its way through the

volumetric dataset and sample values are taken at a defined frequency. The sample values are then mapped into the color space by a so called transfer function (TF) and blended together resulting in the final color for the pixel.

Ray-based approaches for visualizing volumetric datasets have a long tradition [Tuy and Tuy, 1984; Levoy, 1988]. Despite their superior image quality, these approaches have not been suitable for interactive applications for a long time due to their enormous demand for processing power. Often it took hours to compute a single image. Only the emerging, programmable graphics hardware at the beginning of the last decade [Kruger and Westermann, 2003; Roettger et al., 2003] improved the situation. A breakthrough was the introduction of the GeForce 8 series GPUs [NVIDIA, 2006] with its support for executing conditional loops with an arbitrary number of instructions and its tremendous processing power. Nowadays, GPU volume ray-casting allows highly interactive frame rates and has become the preferred method for many applications. It is also the choice for most parts of this thesis and is explained in detail in Section 2.4. The images in Figure 1.1 have been rendered with our IVS volume visualization system using GPU ray-casting.

- **3D Texture Slicing**

  3D texture slicing (Figure 1.2(b)) is a volume visualization method based on 3D texture mapping hardware. In contrast to 2D texture mapping where a two-dimensional texture is projected onto a polygon according to texture coordinates, a three-dimensional texture consists of an entire stack of textures. Defined by three-component texture coordinates, an arbitrary intersection plane through this texture stack can be selected for mapping onto a polygon. Full trilinear filtering is supported by the hardware at near zero cost. For 3D texture slicing, basically the entire volumetric dataset is loaded into such a 3D texture. Depending on the particular method, either a series of axis-aligned or view-aligned slices is then cut out of the 3D texture using texture mapping operations onto rectangular polygons. The final image is synthesized by applying the transfer function and blending of the slices. Especially for applying the transfer function and for blending of the slices a plethora of variations exists with pre-integrated texture slicing [Engel et al., 2001] representing one of the most popular.

  3D texture slicing for volume visualization was developed soon after 3D texture mapping hardware became available [Wilson et al., 1994; Cabral et al., 1994]. For a long time it was the method of choice for interactive volume visualization due to its superior performance, which is mostly limited by the fill rate of the graphics hardware. Disadvantages of the method

are artifacts from the slices, which become visible if the slices are not sampled densely enough (requiring a trade-off with the performance) and non-uniform spacings between the samples if perspective projection is used. However, with the rise of GPU ray-casting, 3D texture slicing has somewhat lost its popularity.

- **Splatting**

  Volume splatting (Figure 1.2(d)) is an object space method. Instead of trying to determine what parts of the volumetric dataset affect a pixel as for example with ray-casting, volume splatting determines which pixels are affected by a discrete element of the volumetric dataset called voxel. Therefore, each voxel of the volumetric dataset is projected onto the image plane and the contribution is added to the affected pixels. Because in most cases the resolution of the volumetric dataset is too coarse for a coherent image and splatting cannot take advantage of inherent trilinear filtering like 3D texture slicing, a so-called interpolation or reconstruction kernel is applied to the voxels before projecting them onto the image plane. In particular, the reconstruction kernel is represented as a 2D footprint, which can be projected onto the image plane by a 2D texture mapping operation [Crawfis and Max, 1993]. In its initial proposition [Westover, 1989], the improper visibility determination of the reconstruction kernels along the z-axis leads to blurry images. Hence, the concept of sheets has been introduced [Westover, 1990; Mueller and Crawfis, 1998; Mueller et al., 1999] where splatting is performed on intermediate sheets, which are blended together for the final image synthesis.

  Splatting is suited for interactive volume visualization. Like with 3D texture slicing, the performance is mostly limited by the fill rate of the graphics hardware. However, the fill rate requirements are typically quite high because each non-empty voxel has to be splatted at least once onto the image plane or sheet. The easy integration of high-order reconstruction kernels with a large support radius is a big advantage of splatting. Also handling large datasets is easy but not necessarily fast as they do not need to fit entirely into the GPU memory. Splatting will be employed for Chapter 5 and is explained in detail in Section 2.5.

- **Shear-Warp**

  As the name already suggests, shear-warp factorization (Figure 1.2(e)) basically consists of a shear step followed by a warp step. For the shear step, the volumetric dataset is subdivided into axis-aligned slices where the axis with the smallest angle to the viewing rays is chosen (assuming orthogonal

projection). These slices are then sheared in the xy-plane in such a way that the viewing rays become perpendicular to the slices. Subsequently the sheared slices are sampled in order to generate an intermediate image. During sampling only bilinear interpolation is applied within the slices but not in between them. Finally, a warp step synthesizes the final image. The warp step is mainly the inverse of the shear transformation in order to get the right orientation and scale. The opacity transfer function can either be run-length encoded in a preprocessing step or is evaluated during rendering at slightly higher cost.

Shear-warp originates from a time [Cameron and Undrill, 1992; Lacroute and Levoy, 1994] where dedicated graphics hardware was not common. Shear-warp targets performance when implemented on the CPU at the price of reduced image quality due to the inaccurate sampling and interpolation. The good performance is partly owed to the favorable memory access patterns of shear-warp. On the other hand it requires storing of three run-length encoded volumes for each of the spatial axes to obtain the slices. Shear-warp may still be a fast algorithm for volume visualization on the CPU but has been superseded by the availability of cheap graphics hardware allowing implementation of different algorithms.

- **Iso-Surface**

  All volume visualization methods presented so far directly visualize the volumetric dataset (Direct Volume Rendering, DVR), enabling easy transfer function changes and display of semi-transparent areas. However, often it is sufficient to visualize a single, opaque iso-surface of the volumetric dataset. Instead of visualizing the iso-surface directly, it can be extracted in the form of a regular triangle mesh and rendered using standard rasterization hardware. The marching cubes algorithm [Lorensen and Cline, 1987] is one of the most popular algorithms for extracting an iso-surface (Figure 1.2(c)). Given an iso-value for extraction, the volumetric dataset is regarded as composed of cubes where each voxel is a vertex of a cube. For each of these cubes and for each edge of a cube, it is determined if the surface intersects the respective edge by comparing the iso-value with the vertex values. If the iso-value lies between the vertex values, the edge is intersected by the surface, otherwise it is not. The actual intersection point is computed by linearly interpolating the vertex values. Knowing which edges of a cube are intersected, the triangle setup for that cube can be obtained by looking up the respective predefined marching cubes case.

  Despite the limitation to a single, opaque iso-surface, iso-surface extraction is still very popular. The triangle rasterization capabilities of today's hard-

ware make the rendering of the already extracted mesh very fast. However, the extraction of the triangle mesh can be expensive, although recent GPU implementations of marching cubes [Dyken et al., 2007] reduce the problem. Iso-surface extraction may also be problematic if the function defining the surface of the selected iso-value is not C^0 continuous.

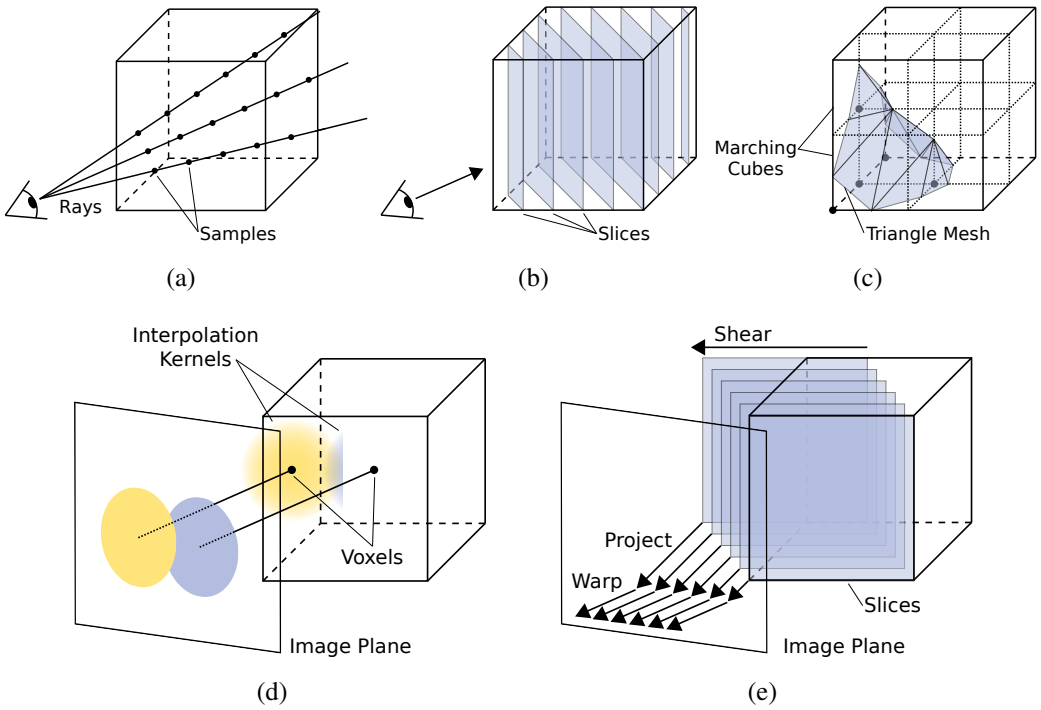


Figure 1.2: *Different methods for volume visualization: volume ray-casting (a), 3D texture slicing (b), iso-surface (c), splatting (d), and shear-warp (e).*

Table 1.1 shows a qualitative analysis of the different methods for volume visualization regarding image quality, performance on current hardware, scalability in terms of achievable performance (not complexity), suitability for high-order interpolation and suitability for displaying transparent areas in combination with opaque surfaces. The comparison represents the general nature of the techniques not taking special enhancements or settings into account. Volume ray-casting on the GPU has become the method of choice as long as high-order interpolation is not necessary due to the superior image quality and the good performance on current hardware.

	Volume Ray-Casting	3D Texture Slicing	Splatting	Shear-Warp	Iso-Surface
Image Quality	+	+-	+	-	+-
Performance	+	+	+-	+-	-
Scalability	+-	+-	-	-	-
High-Order Interpolation	-	-	+	+-	-
Transparency	+	+	+	+	-

Table 1.1: *Comparison of the different methods for volume visualization regarding key aspects.*

1.2 Challenges in Volume Visualization

Interactive volume visualization has become mainstream in recent years and is an important tool in many domains. It is the fact that volumetric datasets can be explored immediately and fully interactively what makes it so compelling. Yet the desire for better image quality, higher resolutions and more features while preserving immediacy and interactivity is a permanent source of challenges. Despite the graphics hardware's amazing rise in performance and capabilities beating Moore's law by far, the most physically accurate algorithms still cannot be executed interactively. Consequently the goal consists of developing algorithms delivering the best possible results in relation to the processing power available. Then again, volume visualization also suffers from functional shortcomings like the time consuming transfer function construction that need to be addressed. A discussion of challenges is presented below and our proposed solutions to these challenges is presented in Section 1.3.

1) **Volume Reconstruction**

   Volumetric datasets are represented as an accumulation of scalar voxel values (in rare cases also vector values), often defined on a regular grid but sometimes also on an irregular grid. In many cases these values are sampled from a scanning device such as a CT scanner or stem from a numerical simulation. By all means it is discrete data with a certain resolution or frequency. As soon as the visualization of such a dataset allows for rotation, zooming and perspective projection, the resolution of the dataset cannot fit the resolution of the output

image in all cases. Often it is too coarse leading to supersampling of the data and the values between the voxels need to be interpolated. But it can also be too fine-grained leading to subsampling of the data and the values need to be interpolated from multiple underlying voxels. In other words, the 3D function defining the scalar field needs to be reconstructed. In their seminal work Marschner and Lobb [Marschner and Lobb, 1994] demonstrated how different reconstruction kernels affect the image quality and how kernels with a large support radius are able to enhance the image quality. The challenge is how to integrate such high-order interpolation kernels with a large support radius into volume visualization while maintaining interactivity. The expensive part is the sampling of a large neighborhood of a voxel to compute the interpolation schema, and therefore, the amount of sampling needs to be limited.

2) **Transfer Function Generation**

Typically volumetric datasets consist of raw scalar data that may represent anything, for example densities, pressures, radiation values, distances or function values. One of the key features of volume visualization is the possibility to explore the dataset by applying a mapping defined by a so-called transfer function (TF) to the raw data before displaying it. In the simplest case this transfer function is a mapping from the space of the raw scalar data to the color and opacity space $f : \mathbb{R} \longrightarrow \mathbb{R}^4$. It allows to specify what parts of the volumetric dataset are opaque, what parts are transparent and what parts are displayed in which color. Setting the opacity mapping of the skin of human body data to transparent enables insight into the body while keeping its contour visible for a better overview. Constructing satisfying transfer functions is one of the most cumbersome tasks in volume visualization. It is time consuming, non-intuitive and the parameters are hard to understand for domain experts not fully familiar with the underlying rendering system. Considering higher dimensional transfer functions where not only the raw scalar data is taken into account but for example the gradient of the dataset as well makes the transfer function construction even harder. Thus, the challenge is to find algorithms for automatically generating transfer functions. Even though some approaches have been suggested [Kindlmann and Durkin, 1998; Fang et al., 1998; Rezk Salama et al., 2006; Zhou and Takatsuka, 2009], mostly targeted to a certain domain, there is a strong need for a well working general purpose method.

3) **Volume Rendering Integral Approximation**

Most volume visualization systems are founded on the volume rendering integral [Moreland, 2004], which is based on the emission and absorption theorem [Max, 1995]. The volume rendering integral basically describes the at-

tenuation of the light emitted by an initial light source on its way through a medium to eye of the viewer. The light is absorbed by tiny particles in the medium but these particles themselves can emit light or reflect light from another light source. Unfortunately, there is no general solution to the volume rendering integral, it cannot be integrated analytically without making confining assumptions. The traditional approach is to use a discretized version of the volume rendering integral by means of a Riemann sum. Further, the part of the volume rendering integral characterizing the attenuation of the light on the remaining way to the eye of the viewer, is usually simplified by developing it into a Taylor series and considering only the first two summands. This has also been justified since it optimally fits the fixed function pipeline of the graphics hardware of the past [Porter and Duff, 1984]. The challenge is to approximate the volume rendering integral as closely as possible. This means to employ an accurate numerical integration scheme, which may not be too expensive and is suited for implementation on the GPU. Also, the approximation of the volume rendering integral can be biased depending on the desired application and different parts of the volume rendering integral can be approximated with different accuracy.

4) **Illumination Model**

The illumination model plays an important role for the perception of the images generated by a volume visualization system. It is the shading from the diffuse reflections accentuating the depth perception and the specular highlights creating the impression of a glossy surface. The domain of computer graphics has a long history of developing lighting and shading models, often enough driven by the limited processing power of the hardware of the past. One of the most popular shading models is the Blinn-Phong model [Phong, 1973; Blinn, 1977], consisting of an omnipresent ambient light term, a diffuse reflection term and a specular reflection term. It does not take shadows or global illumination effects into account, albeit shadows and global illumination effects are crucial for generating realistic looking images and also for a good depth perception. On the other hand, sophisticated illumination models such as global illumination [Veach and Guibas, 1997; Jensen, 2001] or radiosity [Cohen and Greenberg, 1985; Wallace et al., 1987] have been proposed a long time ago. Unfortunately, the evaluation of these models is prohibitively expensive for interactive applications. The challenge consists of implementing similar effects in volume visualization systems maintaining interactivity. These effects don't have to hold complete physical accuracy but they need to look physically plausible and convincing. Often the goal is to find the right simplifications making it fast while not affecting the subjective perception of the effect.

5) **Performance**

Moore's law stating that the complexity (and thus performance) of CPUs doubles every 18 months has been amazingly accurate for the last 30 years. Graphics hardware evolving into programmable GPUs was even capable of doubling the complexity at a much faster rate over a long period of time. Despite this tremendous progress making many things possible thought impossible, performance is still a ubiquitous concern. Given the condition that volume visualization must be interactive, it is always a trade-off between image quality, performance and features competing for the valuable resource of computation time. The challenge is to find more efficient algorithms requiring less computation time while maintaining a good image quality at the same time. Always this comes at the price of a compromise, and the challenge is figuring out the right compromise with the least impact. Examples of such compromises are level-of-detail (LOD) or adaptive sampling methods to avoid spending much computation time in areas where it makes hardly a difference regarding image quality.

1.3 Contributions

In this thesis we elaborate on solutions for some of the major challenges raised in Section 1.2. First, we address one of the most cumbersome tasks in volume visualization, the construction of transfer functions. Notably, we propose a new general purpose algorithm for automatically generating transfer functions based on information theory. Second, we focus on realism and depth perception of GPU volume ray-casting. We show how realism and depth perception can be augmented by developing an illumination model based on a special representation of the volume rendering integral. Third, we study reconstruction kernels in volume splatting in terms of quality and performance. We suggest a method for boosting the performance of volume splatting including a special reconstruction kernel. A schematically organized overview of our contributions is given in Figure 1.3.

In the following paragraphs our contributions are discussed in more detail with regard to the functional and non-functional challenges.

1) *Transfer Function Generation*
Visibility-Difference Entropy Transfer Function Generation

We propose a method for automatically generating transfer functions based on information theory. Unlike other methods, targeting specific domains or selected criteria only, our method is more general purpose and can be used for any dataset. We achieve this by presenting the user an entire set of transfer functions producing the images with the highest information content according

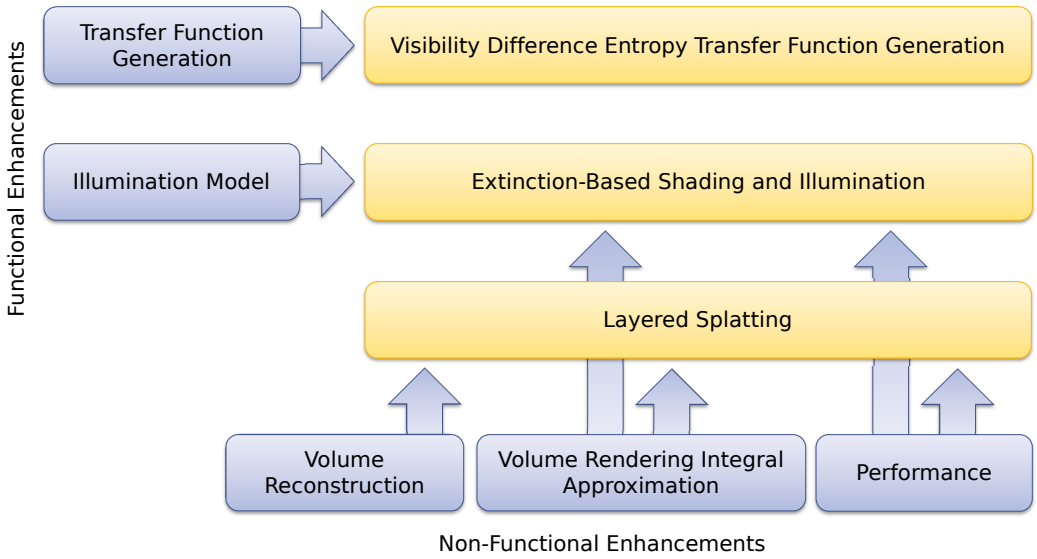


Figure 1.3: *In this thesis we address some of the major functional and non-functional challenges of volume visualization. We introduce a general purpose method for automatically generating transfer functions, we present a new, fast illumination model and we boost the performance of volume splatting.*

to a newly defined metric. Often these images are completely different from each other allowing insight into different aspects of the dataset. The advantage is that the user gets an overview of the characteristics of the dataset without any previous knowledge and may continue with the most suitable transfer functions.

The basic idea is that the user chooses a basis function followed by a search in the parametric space of that basis function. For each parameter set a corresponding transfer function is generated and scored by the information content of the resulting images. The transfer functions reaching the highest scores are then presented to the user. By choosing different basis functions and search methods, the user can influence the transfer function generation regarding the properties of the transfer functions and the time required for the search. Depending on the search method, the user is able to obtain representative results in a few minutes.

2) *Illumination Model, Volume Rendering Integral Approximation, Performance*
Extinction-Based Shading and Illumination in GPU Volume Ray-Casting

We augment realism and depth perception of GPU volume ray-casting by introducing an extinction-based, unified illumination model. Our illumination

model enables directional soft shadows taking scattering into account, ambient occlusion and color bleeding. It fully supports multiple dynamic point, spot and area lights as well as light sources residing inside the dataset. No preprocessing is required and interactive transfer function changes are possible. While greatly enhancing realism and depth perception, the performance of our implementation is very competitive, not harming interactivity or responsivity.

Commonly, the volume rendering integral is based on a discretized version of the original volume rendering integral. Specifically, the original exponential extinction coefficient is approximated by the first two summands of its Taylor series expansion resulting in α-blending. In contrast to α-blending forming a product when sampling along a ray, the original exponential extinction coefficient is an integral and its discretization a Riemann sum. Our illumination model is cleverly built upon the fact that it is a sum, which can be computed very efficiently on the GPU.

3) *Performance, Reconstruction, Volume Rendering Integral Approximation*
Layered Splatting

We enhance volume splatting by suggesting hardware-accelerated *layered splatting*. Layered splatting is able to boost the performance compared to prior state-of-the-art high-quality splatting methods. It incorporates a special cubic reconstruction kernel making an optimal compromise between image quality and the required support radius. Further, the image quality is enhanced by choosing the same approximation of the volume rendering integral as with the extinction-based illumination model. We show that using a Riemann sum discretization instead of the Taylor series expansion for the extinction coefficient yields a better image quality.

To benefit from the superior image quality provided by reconstruction kernels with large support radii, the visibility determination along the axis perpendicular to the sheets must be as accurate as possible. In sheet-buffer splatting this is achieved by cutting the reconstruction kernel into several slices and splat each slice onto the closest sheet quickly reaching the rasterization bound of the graphics hardware. The basic idea of layered splatting is to reduce the required splatting operations by a smart combination of the cubic reconstruction kernel in combination with virtual splatting operations.

1.4 Dissertation Overview

Chapter 2 delivers the basic knowledge about volume rendering in general and volume rendering on the GPU in particular. It starts with the theory on reconstruction of the volumetric dataset including the theoretical limits and an overview

of different reconstruction filters. Given the reconstructed dataset, it is described how the values can be mapped into the color/opacity space by one- and multi-dimensional transfer functions. The foundation of volume rendering in terms of the volume rendering integral with its possible approximations is explained subsequently. Further, we show how volume ray-casting and splatting work and how they can be implemented on the GPU.

Chapter 3 introduces our visibility-difference entropy transfer function generation. The focus lies on the major parts of the system, namely the definition of the information theory-based metrics for scoring transfer functions, the range of basis functions and the different search strategies. The suitability of the method is evaluated by applying it to a variety of datasets and showing how well the characteristics of the datasets are represented by the results. Finally, a series of pictures illustrates the coherence of the results together with the respective timings.

Chapter 4 presents our extinction-based illumination model for GPU ray-casting. We show the weaknesses of the common Blinn-Phong model and how soft directional shadows, ambient occlusion and color bleeding can augment realism and depth perception. Special attention is paid to the approximation of light scattering effects. Further, the efficient implementation - prerequisite for interactive frame rates and responsivity - is explained in-depth. Finally, the results are illustrated by a range of representative images and detailed timings.

Chapter 5 explains our approach to layered splatting. It is discussed where the performance limitations of state-of-the-art sheet-buffer splatting lie. Knowing the reasons for the performance limitations, we highlight the boost achievable by the smart combination of a special reconstruction kernel in combination with virtual splatting operations. In addition, the quality gains by the closer approximation of the original extinction coefficient of the volume rendering integral are investigated and illustrated.

Chapter 6 concludes this dissertation with a summary of the work and motivation for future work.

CHAPTER 2

GPU-BASED DIRECT VOLUME VISUALIZATION

2.1 Volume Reconstruction

Volumetric datasets used in volume visualization are typically acquired by a sampling process. This sampling process can be carried out either through a scanning device like a CT, MRI, PET or ultrasound scanner or by storing results from a numerical simulation. By all means it is discrete scalar or sometimes vector data stored on a regular or irregular grid. Throughout this thesis only scalar data defined on a regular grid is considered but everything discussed basically applies to vector data or irregular grids as well.

Assuming a continuous scalar function $f : \mathbb{R}^3 \longrightarrow \mathbb{R}, (x, y, z) \longmapsto f(x, y, z)$, the sampled data is defined as:

$$g(x, y, z) := f(x, y, z)\Delta(x, y, z) \tag{2.1}$$

where

$$\Delta(x, y, z) := \sum_{i,j,k \in D} \delta(x - iT, y - jT, z - kT). \tag{2.2}$$

T is the sampling interval, $D \subset \mathbb{Z}$ is the sampling range, which can also be different for the three spatial axes, and δ is the Dirac delta function. Consequently g is zero outside of the sampling range or when the position (x, y, z) is not a

whole-number multiple of T. The volumetric dataset V is then defined as:

$$V := \{g(iT, jT, kT) \mid i, j, k \in D\}. \tag{2.3}$$

At some point all volume visualization methods discussed in Chapter 1 re-sample the volumetric dataset. In particular, if volume ray-casting is used, view rays are shot from the eye of the viewer through the volumetric dataset and samples are taken along these rays. The positions of the samples on the rays will hardly ever match the positions of existing sample values (voxels) in the volumetric dataset as illustrated in Figure 2.1. Thus, the value for a desired position needs to be interpolated from the surrounding voxels or more precisely, f needs to be reconstructed from g. Unfortunately a perfect reconstruction of f from g is not possible as discussed below.

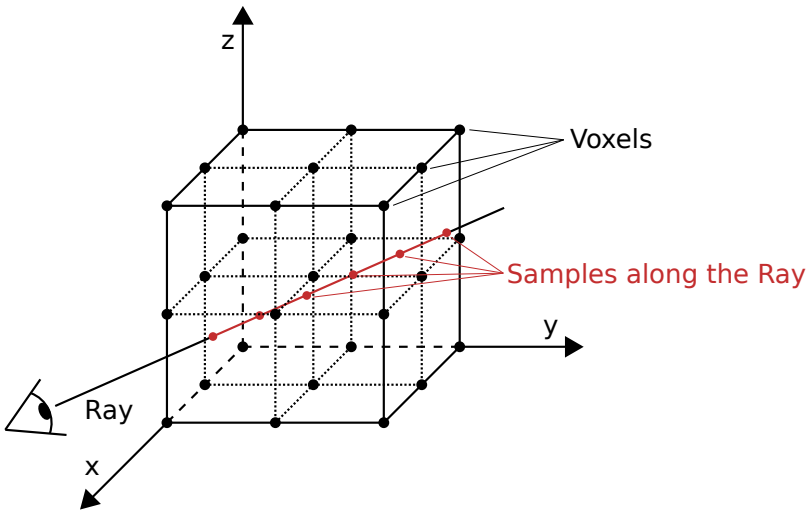


Figure 2.1: *A ray is shot through a volumetric dataset consisting of discrete voxels and samples along the ray are taken. Because the positions of the samples on the ray do not match the voxel positions from the dataset, the values of the samples have to be interpolated from the surrounding voxels.*

2.1.1 Frequency Domain

Understanding why a perfect reconstruction of f from g is not possible requires a look at the sampling process in the frequency domain [Gonzalez and Woods, 2006; Foley et al., 1990; Marschner and Lobb, 1994]. If not otherwise stated, the formalism from Marschner and Lobb is used for frequency domain considerations. The Fourier transform, named after the famous mathematician [Fourier, 1822],

describes the frequency spectrum of a signal. Considering the scalar function f to be a continuous signal, its Fourier transform F is defined as:

$$F(\omega_x, \omega_y, \omega_z) := \int_{\mathbb{R}^3} f(x, y, z) e^{-i(\omega_x x + \omega_y y + \omega_z z)} dx dy dz. \tag{2.4}$$

Having the Fourier transform F, the underlying signal f can be obtained by:

$$f(x, y, z) := \frac{1}{(2\pi)^3} \int_{\mathbb{R}^3} F(\omega_x, \omega_y, \omega_z) e^{i(\omega_x x + \omega_y y + \omega_z z)} d\omega_x d\omega_y d\omega_z. \tag{2.5}$$

For the remainder of the section all lowercase letters represent functions in the *spatial domain* and all capital letters Fourier transforms in the *frequency domain*.

The convolution theorem [Gonzalez and Woods, 2006] states that multiplying two functions in the spatial domain, in particular f and h from the sampling process in Equation 2.1, is equivalent to convolving the respective Fourier transforms F and H in the frequency domain. This is also valid in the other direction: Multiplying F and H in the frequency domain is equivalent to convolving f and h in the spatial domain.

Given that h is a grid of Dirac impulses, it can be shown that its Fourier transform H is an impulse grid as well. Hence, convolving F with H yields a copy of the frequency spectrum F over each impulse of H. Of special interest is the spectrum at the origin of the frequency domain called *primary spectrum*. All the other spectra are called *alias spectra* and are undesired for reconstruction. The entire process is demonstrated in Figure 2.2 with the help of a one-dimensional function.

2.1.2 Perfect Reconstruction

In Figure 2.2 the spectra for G do not overlap but are nicely arranged. This, however, is not always the case and the spectra may start overlapping if the function f contains frequencies greater than half the sampling frequency ω_N for the respective axis. ω_N is called the Nyquist frequency according to the Nyquist-Shannon sampling theorem [Nyquist, 1928; Shannon, 1949] and is the limiting frequency for non-overlapping spectra. Therefore, to avoid overlapping spectra, f needs to be band-limited.

If f is properly band-limited such that the spectra do not overlap, F can be perfectly reconstructed by multiplying G with an ideal low-pass or so-called reconstruction filter S:

$$S(\omega_x, \omega_y, \omega_z) := \begin{cases} 1, & \text{if } \omega_x, \omega_y, \omega_z \in [-\omega_{N_{x,y,z}}, \omega_{N_{x,y,z}}] \\ 0, & \text{otherwise} \end{cases}. \tag{2.6}$$

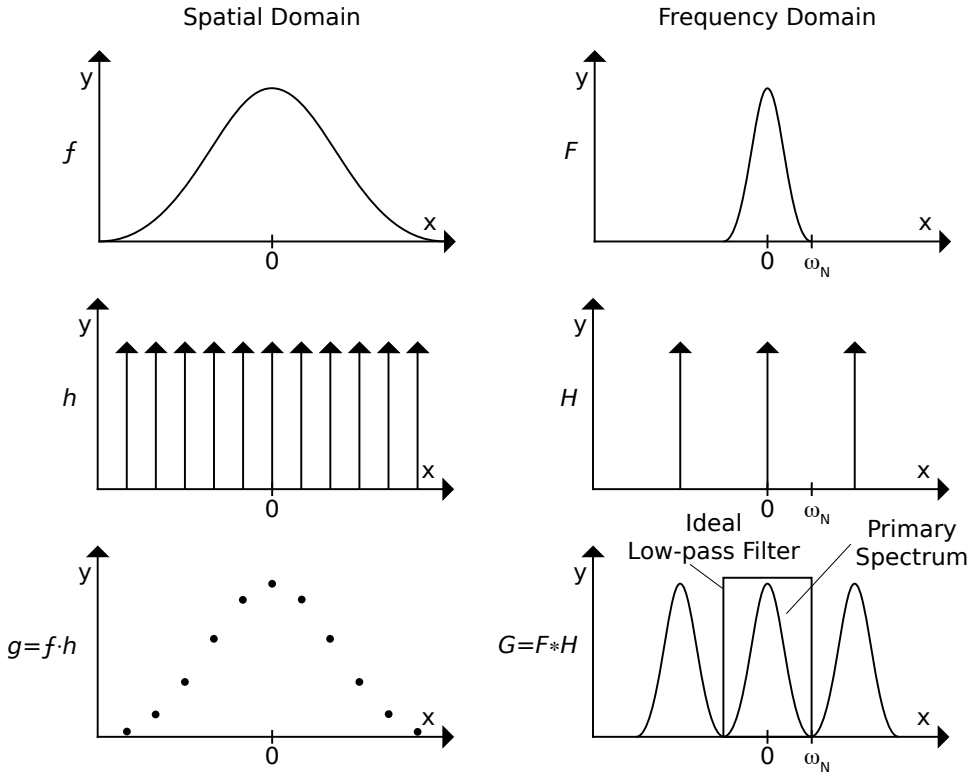


Figure 2.2: *The sampling process in the spatial and in the frequency domain.*

S is a cube that filters out all alias spectra retaining the primary spectrum only, which is the reason why it is called *reconstruction filter*. It is a low-pass filter since the primary spectrum is located at the origin of the frequency domain.

Multiplying G with S in the frequency domain is equivalent of convolving g with s in the spatial domain where s is the inverse Fourier transform of S:

$$s(x, y, z) := (2\omega_N)^3 \operatorname{sinc}(2\omega_N x) \operatorname{sinc}(2\omega_N y) \operatorname{sinc}(2\omega_N z). \tag{2.7}$$

Thus, convolving g with s is the perfect reconstruction of f but unfortunately it is not possible in practice due to the infinite extent of s.

2.1.3 Reconstruction Filters

Since perfect reconstruction of f from g is not possible in practice, imperfect reconstruction filters are employed. There are several classes of imperfect reconstruction filters and each filter has its own advantages and disadvantages. Surveys of reconstruction filters with focus on computer graphics and volume visualization are numerous [Mitchell and Netravali, 1988; Westover, 1990; Carlbom, 1993;

Marschner and Lobb, 1994]. Marschner and Lobb suggest the following classification of reconstruction filters and artifacts:

- **Separable Filters**

  Reconstruction filters of this class can be separated into independent one-dimensional filters for each spatial axis:

  $$s(x, y, z) := s_s(x)s_s(y)s_s(z). \tag{2.8}$$

  Included in this class are some of the most popular reconstruction filters like trilinear interpolation

  $$s_s(x) := 1 - |x| \text{ for } |x| < 1, \tag{2.9}$$

  cubic filters where $s_s(x)$ is a polynomial with $\deg(s) = 3$ including B-splines, truncated Gaussian filters

  $$s_s(x; \sigma, m) := e^{-\frac{x^2}{2\sigma^2}} \text{ for } |x| < m, \tag{2.10}$$

  and also windowed sinc filters

  $$s_s(x; m) := w(x; m)\,\mathrm{sinc}(x). \tag{2.11}$$

  Windowed sinc filters try to approximate the ideal reconstruction filter by limiting its infinite support. Artifacts occurring as a result of the truncated support are weakened by a smooth drop-off window w [Theussl et al., 2000].

- **Spherically Symmetric Filters**

  Spherically symmetric filters have only a single parameter, namely the distance from the origin and can be expressed as:

  $$s(x, y, z) := s_r(\sqrt{x^2 + y^2 + z^2}). \tag{2.12}$$

  This class includes spherically symmetric versions of windowed sinc filters and often spherically symmetric versions of the Gaussian filter:

  $$s(r; m) := e^{-\frac{r^2}{2\sigma^2}} \text{ for } r < m. \tag{2.13}$$

  One thing making spherically symmetric filters attractive - especially for splatting algorithms - is their isotropic characteristic.

- **Pass-band Optimal Filters**

  Pass-band optimal filters [Hsu and Marzetta, 1989] are separable filters adapted for volume visualization by Carlbom [Carlbom, 1993]. The basic idea is to regard the construction of a filter as a minimization problem using a weighted Chebyshev minimization schema to solve it. The focus of these filters lies on minimizing smoothing.

None of the filters above is a perfect cube in the frequency domain and therefore none of these filters perfectly cuts out the primary spectrum. As a result all filters suffer from artifacts under certain conditions, though the type of artifacts and the conditions when artifacts occur are different. Assuming that the original signal f is properly band-limited and that the sampling frequency is at least twice ω_N, there remain aliasing and smoothing artifacts solely attributed to the imperfect reconstruction. The third kind of artifact called ringing originates from discontinuities in f but is also influenced by the reconstruction filter.

- **Aliasing**

  Aliasing [Mertz and Gray, 1934; Foley et al., 1990] occurs if the primary spectrum is not properly separated from the alias spectra in the frequency domain. This is either the case if the primary spectrum overlaps with the alias spectra (pre-aliasing) or if the reconstruction filter does not properly cut-off at ω_N but extends into the alias spectra (post-aliasing). In either case parts of the frequency spectrum of the original signal occur at different frequencies in the reconstructed signal as aliases.

- **Smoothing or Blurring**

  In a way smoothing is the contrary of aliasing. It occurs if the reconstruction filter starts to cut off before ω_N and parts of the primary spectrum get lost or are averaged. This is typically the case for the higher frequencies representing fine structure in a volumetric dataset. This fine structure is lost or occurs blurred in the reconstructed dataset.

- **Ringing**

  Ringing [McGillem and Cooper, 1984] is a type of artifact that occurs if the original signal contains discontinuities. It is visible in the reconstructed signal just around the discontinuities as ringing waves. Because ringing is not the result of an imperfect reconstruction but a peculiarity of Fourier series at discontinuities, the ringing would be retained even with a perfect reconstruction filter. Conversely ringing can be extenuated by using an imperfect reconstruction filter containing smoothing.

Choosing the right reconstruction filter for reconstructing the volumetric dataset is always a trade off between the advantages and disadvantages of the different filters. Apart from the quality a reconstruction filter provides, the cost of evaluating it plays an important part considering interactive volume visualization. This is also the reason why many volume visualization systems use trilinear interpolation as a filter since it is built into the hardware and comes basically at zero cost.

2.2 Transfer Function

One of the stunning features of direct volume visualization is the possibility to specify what parts of a volumetric dataset are displayed with a particular color and opacity. Important parts can be highlighted with a salient color and uninteresting parts made entirely transparent. Structures not in focus but helpful for supporting the overall perception may be presented semi-transparent (see Figure 2.3). This opens the opportunity to gain different insights into a volumetric dataset and is of special importance if only selected parts need to be investigated (i.e. a medical doctor investigating a CT scan, see also Figure 1.1).

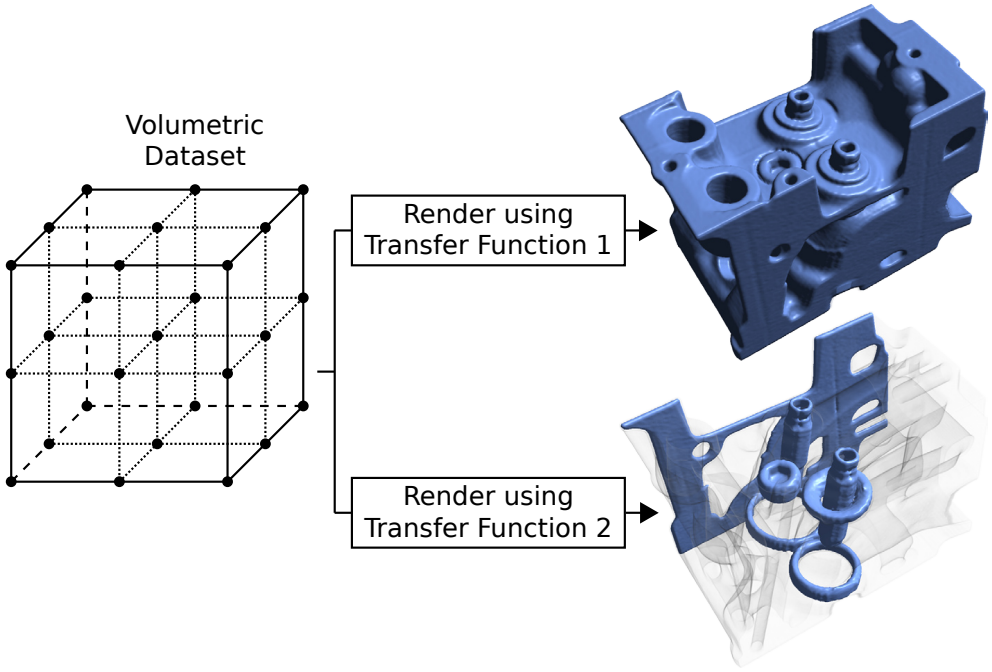


Figure 2.3: *Visualization of a volumetric dataset containing a cylinder head with different transfer functions. For transfer function 1 the focus lies on the entire block and for transfer function 2 the focus is on the valve guides showing the block semi-transparent. Dataset courtesy of General Electric.*

The specification of how to map the scalar values from the volumetric dataset into the color and opacity space is commonly called *transfer function (TF)*. In fact, in the vast majority of cases specifying a transfer function is not only a feature but required since the scalar data from the volumetric dataset has no inherent meaning in the color and opacity space. For example, it is not obvious how the radiodensity values of a CT scan should look like in the color and opacity space. Then again, if no explicit restrictions are imposed, the number of degrees of freedom of a transfer function is typically large. This makes the manual construction of a transfer function very time consuming and also unintuitive, requiring an automated solution.

2.2.1 One-Dimensional Transfer Function

The simplest case of a transfer function is a one-dimensional transfer function:

$$t : \mathbb{R} \longrightarrow \mathbb{R}^4, x \longmapsto t(x). \tag{2.14}$$

It is only dependent on a single parameter and returns a four-tuple (R, G, B, A) containing the red, green, blue and α portion (opacity) for the respective input value. In most cases the input is the scalar value from the volumetric dataset: $t(f(x, y, z))$. However, in some special applications the input may also be the first derivative of the scalar value $t(f'(x, y, z))$, highlighting rapid changes in the dataset only. In fact, a multitude of possible input parameters is thinkable but in practice parameters other than f are rather used as additional dimensions as shown in the next subsection. Simple examples of one-dimensional transfer functions are demonstrated in Figure 2.4.

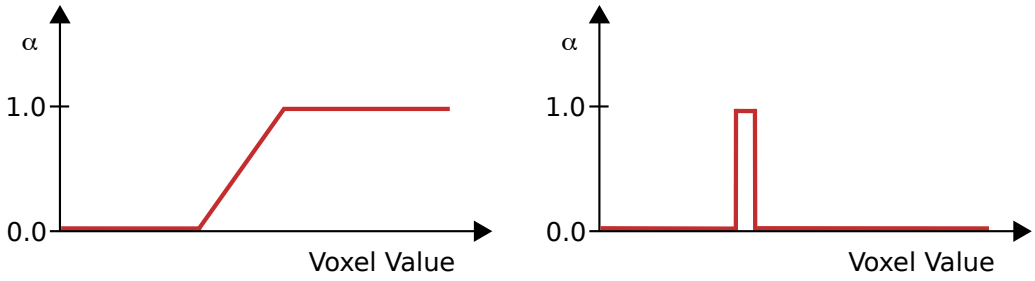


Figure 2.4: *Simple examples of one-dimensional transfer functions showing the α-part only. On the left the transfer function is ramp-shaped representing a common family of transfer functions and on the right only a very small range of values is taken into account representing an iso-surface.*

In an actual implementation, the definition space of t is limited to a certain range and precision, not requiring t to be a continuous, analytical function. Often t

is specified by the user drawing a curve, which is then stored as a discrete mapping table.

All transfer functions generated in Chapter 3 as well as the transfer functions used in Chapters 4 and 5 are one-dimensional, taking scalar values from the volumetric dataset as input. Technically, the transfer functions are stored as 8 bit discrete mapping tables.

2.2.2 Multi-Dimensional Transfer Function

A drawback of one-dimensional transfer functions is that regions of the volumetric dataset comprising equal values cannot be differentiated. One can imagine a volumetric dataset where the values represent densities. Even if there are two different structures present made out of different materials, they cannot be differentiated if they have by chance the same density. To solve the problem, multi-dimensional transfer functions can be employed:

$$t : \mathbb{R}^n \longrightarrow \mathbb{R}^4, x_1, \ldots, x_n \longmapsto t(x_1, \ldots, x_n). \tag{2.15}$$

Very popular is the two-dimensional transfer function having the scalar value from the volumetric dataset as the first dimension and the first derivative of this scalar value as the second dimension as introduced by Levoy [Levoy, 1988]. This additional dimension allows for emphasizing complex material boundaries, which is not possible by just considering the scalar value. The gradient of the scalar value $\nabla f(x, y, z)$ is taken as an approximation of the derivative and the magnitude of the gradient $\|\nabla f(x, y, z)\|$ is used as the second dimension for the transfer function: $t(f(x, y, z), \|\nabla f(x, y, z)\|)$.

For an even more precise disambiguation of complex material boundaries the second derivative can be used as a third dimension [Kniss et al., 2001] but also completely different approaches have been presented. Takeshima et al. [Takeshima et al., 2005] use topological attributes computed from the level-set graph of a volumetric dataset as a dimension. Especially useful for medical applications are size-based transfer functions [Correa and Ma, 2008] taking the size of a local feature for a voxel into account. This makes it possible to highlight structures of a certain size, for example for detecting a blastoma. Also interesting for medical applications are texture-based transfer functions [Caban and Rheingans, 2008] taking the local texture properties of a voxel into account. For example, this allows to differentiate between blood vessels and the boundary of the lungs even if they have similar scalar values in the volumetric dataset. A survey of transfer functions suitable for volume visualization can be found in this work [Arens and Domik, 2010].

We do not use multi-dimensional transfer functions in this thesis. Extending visibility-difference entropy transfer function generation from Chapter 3 for

multi-dimensional transfer functions is possible, but would require dealing with the additional dimensions. The methods in Chapters 4 and 5 work independently of the type of transfer functions and can handle multi-dimensional transfer functions out-of-the-box.

2.2.3 Segmentation

The segmentation of a volumetric dataset is important and frequently used although not directly related to transfer functions. Segmentation [Udupa and Herman, 1999] describes the process of assigning the voxels of a volumetric dataset to different segments employing an appropriate technique. For visualization purposes, each segment can have its own transfer function. Determining the segment a voxel belongs to allows to pick the respective transfer function for that segment during visualization.

However, the chance of treating different voxel segments independently requires the segments to be created in the first place. It is a research area on its own how to manually and automatically create good segmentations.

2.3 Volume Rendering Integral

If volume rendering techniques such as ray-casting or volume splatting are used for volume visualization, an underlying model is required. This model establishes a theoretical basis for generating the color of a pixel in the final image from the volumetric data. Yet the way the model is implemented in terms of an algorithm is not inherent. Many volume rendering algorithms are founded on the volume rendering integral as their model, which will be explained in this section. An in-depth discussion of the volume rendering integral including variations and limitations is provided in this work [Moreland, 2004]. Apart from the volume rendering integral many other models exist such as maximum intensity projection (MIP) [Wallis et al., 1989] (called maximum-activity projection in this seminal paper) but these models are not further discussed in this thesis.

The volume rendering integral originates from Kajiya and von Herzen [Kajiya and von Herzen, 1984] but is more formally developed from the emission and absorption theorem by Max [Max, 1995]. Consider a scenario where at a certain spatial position 0 light with intensity I_0 is emitted towards the eye of a viewer who is located at a different spatial position D. On the way from the source to the eye of the viewer, the light travels through a medium filled with tiny particles. When the light hits these tiny particles, parts of it can be scattered in different directions or can be absorbed by the tiny particles. Thus, the overall light intensity decreases. On the other hand, these tiny particles can scatter incoming light from another

light source towards the eye of the viewer or they can also emit light themselves. Hence, the overall light intensity increases.

If not otherwise stated, the formalism from Max [Max, 1995] is used to quantify the volume rendering integral. Instead of examining an open space between the light source and the eye of the viewer, a cylinder is assumed as an auxiliary construct (Figure 2.5). The light then travels through this cylinder on the way from the source to the eye of the viewer. Consider a slab of the cylinder with cross-sectional area E and thickness Δs. Given the density of the tiny particles in the cylinder ρ with a cross-sectional area A per particle, the area covered by the particles in the slab is $\rho AE\Delta s$. If Δs is small enough (tending to zero) such that the particles do not overlap, the fraction of occluded light at the base of the slab amounts to:

$$\rho AE\Delta s/E = \rho A\Delta s. \tag{2.16}$$

If the tiny particles emit themselves light with an intensity C per unit area, the amount of additional light can be deduced similarly to the occluding part: $C\rho AE\Delta s$. Accordingly the fraction of additional light at the base of the slab amounts to:

$$C\rho AE\Delta s/E = C\rho A\Delta s. \tag{2.17}$$

It is now possible to formulate the variation of the light intensity I by the following differential equation:

$$\frac{dI}{ds} = C(s)\tau(s) - I(s)\tau(s) \tag{2.18}$$

where $\tau(s) = \rho(s)A$ is called the *extinction coefficient* reflecting the amount of absorbed light. For a detailed description on how to solve Equation 2.18 we refer to Max [Max, 1995]. With $e^{-\int \tau(t)dt}$ as approach the solution yields:

$$I(D) = I_0 e^{-\int_0^D \tau(t)dt} + \int_0^D C(s)\tau(s)e^{-\int_s^D \tau(t)dt}ds. \tag{2.19}$$

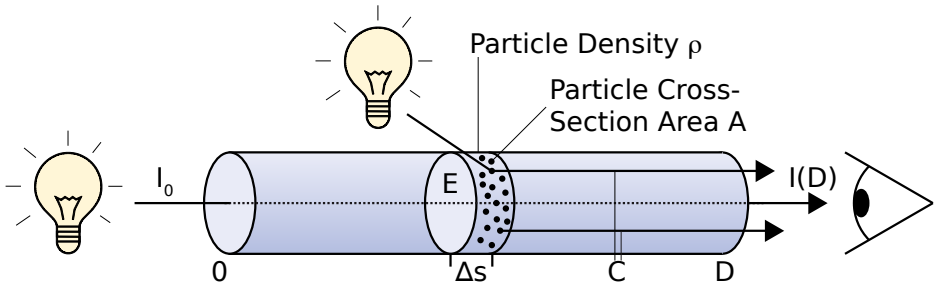


Figure 2.5: *Light with intensity I_0 travels from the light source at position 0 to the eye of the viewer at position D through a cylinder filled with tiny particles. The tiny particles absorb light, reflect light from another light source or emit light themselves.*

This integral is known as the *volume rendering integral*. It describes the exponential extinction of the initial light intensity I_0 on the way from the source to the eye of the viewer. Further, it integrates over the light emitted by the tiny particles on this way where the emitted light itself is subject to the exponential extinction on the remaining way to the eye of the viewer.

The emitted light of the tiny particles denoted by $C(s)\tau(s)$ deserves a closer look. In many implementations $C(s)\tau(s)$ is substituted by a simple and cheap function depending on the value at the particular spatial position only. With such a simple function however, it is not possible to model the phenomenon called scattering. Scattering (Figure 2.6), especially visible in high albedo media, describes the reflection of parts of a light ray in more or less random directions when the ray hits a tiny particle. This means that for a certain position not only the light on a straight trajectory from the light source to the eye of the viewer has to be taken into account but also light being scattered into the direction of the eye by the tiny particles at that position. Again we use the formalism from Max [Max, 1995] to quantify the model. The scattered light S for a position x and a direction υ is defined as:

$$S(x, \upsilon) = R(x, \upsilon, \upsilon')I(x, \upsilon') \tag{2.20}$$

where $I(x, \upsilon')$ is the light reaching x from direction υ' and $R(x, \upsilon, \upsilon')$ is a bidirectional reflection distribution function (BRDF). R basically expresses the fraction of light reflected into direction υ coming from direction υ' at position x. Composing R of a particle albedo factor, a particle scattering rate, and a Henyey-Greenstein function [Henyey and Greenstein, 1941] is a possible variation. In high albedo media or if the particle density is high, it is likely that a light ray is scattered multiple times. Therefore, all incoming directions υ' have to be taken into account and S needs to be integrated over the unit sphere Ω:

$$S(x, \upsilon) = \int_{\Omega} R(x, \upsilon, \upsilon')I(x, \upsilon')d\upsilon' \tag{2.21}$$

More generally, the volume rendering integral taking scattering into account can be written as:

$$I(D) = I_0 e^{-\int_0^D \tau(t)dt} + \int_0^D (E(s) + S(s, \upsilon))e^{-\int_s^D \tau(t)dt} ds \tag{2.22}$$

where $E(s)$ is the self-emission of the particles. In Chapter 4 we develop a fast illumination and shadowing model based on an innovative approximation of Equation 2.22.

2.3.1 Riemann Sum

So far, the volume rendering integral has been presented in its continuous form. Unfortunately, an analytical solution is only possible assuming restrictive bound-

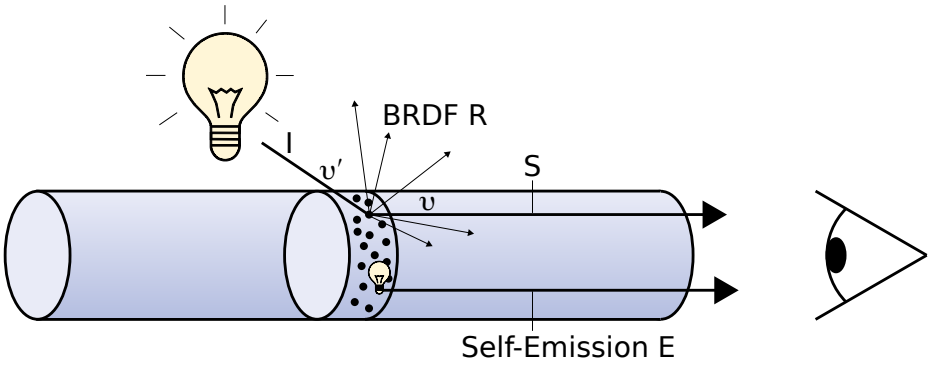


Figure 2.6: *Light with intensity I coming from direction v' hits a particle and is scattered according to a bidirectional reflection distribution function (BRDF) R. The intensity of the scattered light in direction v towards the eye of the viewer amounts to S. Another particle emits light with intensity E itself.*

ary conditions [Moreland, 2004]. To avoid this, the volume rendering integral is usually developed into a Riemann sum and integrated piecewise. If not otherwise stated, the formalism from Moreland [Moreland, 2004] is used.

Starting with the volume rendering integral from Equation 2.19, the exponential extinction term $e^{-\int_0^D \tau(t)dt}$ can be discretized as

$$e^{-\sum_{i=0}^{D/\Delta t} \tau(t_i)\Delta t} = \prod_{i=0}^{D/\Delta t} e^{-\tau(t_i)\Delta t} \tag{2.23}$$

where Δt is the step size. Similarly discretizing the outer integral results in the following approximation for the volume rendering integral:

$$I(D) \approx I_0 \prod_{i=0}^{D/\Delta t} e^{-\tau(t_i)\Delta t} + \sum_{i=0}^{D/\Delta t} C_i \tau(t_i)\Delta t \prod_{j=i+1}^{D/\Delta t} e^{-\tau(t_j)\Delta t}. \tag{2.24}$$

It is now in a discretized form such that it can be integrated piecewise. Though, for typical implementations the emission intensity C and the extinction coefficient τ are often substituted by the color from the transfer function and an opacity value α as derived below. This makes it much easier because typical graphics hardware is optimized for working with color and α-values rather than emission intensities and extinction coefficients.

2.3.2 Alpha Blending

In a next step the extinction term is further simplified eventually leading to α-blending. Developing $e^{-\tau(t_i)\Delta t}$ for a given i into a Taylor series around zero and

taking the first two elements yields:

$$e^{-\tau(t_i)\Delta t} \approx 1 - \tau(t_i)\Delta t. \tag{2.25}$$

Setting $\tau(t_i)\Delta t = \alpha_i$ and additionally assuming that the emitted intensity C_i is the color of a voxel leads to the final equation:

$$I(D) \approx I_0 \prod_{i=0}^{D/\Delta t} (1 - \alpha_i) + \sum_{i=0}^{D/\Delta t} C_i \alpha_i \prod_{j=i+1}^{D/\Delta t} (1 - \alpha_j). \tag{2.26}$$

The attenuation $\prod (1 - \alpha)$ from Equation 2.26 is equal to the over or under operator for back-to-front or front-to-back compositing respectively from Porter and Duff [Porter and Duff, 1984]. This operator, commonly known as α-blending, has been implemented in graphics hardware for a very long time and comes at near zero cost regarding performance. This is also the reason why many volume rendering algorithms are founded on exactly this version of the volume rendering integral.

2.3.3 Exponential Extinction

Nowadays the powerful, programmable GPUs permit implementing arbitrary compositing and blending algorithms retaining a competitive performance. Since a few years this is also valid if floating point arithmetic is used instead of fixed point arithmetic. Consequently we promote going one step back to Equation 2.24 and use the original exponential extinction term in place of the linear attenuation from Equation 2.26:

$$I(D) \approx I_0 \prod_{i=0}^{D/\Delta t} e^{-\tau(t_i)\Delta t} + \sum_{i=0}^{D/\Delta t} C_i \tau(t_i)\Delta t \prod_{j=i+1}^{D/\Delta t} e^{-\tau(t_j)\Delta t}. \tag{2.27}$$

Instead of regarding the extinction term as a product of individual exponential coefficients, we regard the extinction term as the exponential of a sum of extinction summands effectively reverting the simplification from Equation 2.23:

$$I(D) \approx I_0 e^{-\sum_{i=0}^{D/\Delta t} \tau(t_i)\Delta t} + \sum_{i=0}^{D/\Delta t} C_i \tau(t_i)\Delta t e^{-\sum_{j=i+1}^{D/\Delta t} \tau(t_j)\Delta t}. \tag{2.28}$$

Replacing α by the original extinction coefficient τ imposes a new transfer function design. In practical implementations α is the linear opacity with a definition interval $[0, 1]$ whereas τ is used in the exponential extinction term and implies a theoretical definition interval $[0, \infty]$. If α is used as τ the absolute error between

the two different extinction terms $|\alpha + e^{-\alpha} - 1|$ is negligible for small (transparent) values of α only. For α values near 1 representing opaque regions the difference is substantial and thus existing transfer functions based on α need to be transformed:

$$\alpha \longmapsto \tau = (1 - e^{-\alpha})^{-1} = \ln\left(\frac{1}{1-\alpha}\right). \tag{2.29}$$

In Chapter 4 we demonstrate how the fact that the extinction term now consists of the exponential of an order-independent sum can be exploited for our novel illumination model. In Chapter 5 we evaluate the quality improvements resulting from the closer approximation of the volume rendering integral.

2.4 GPU Ray-Casting

The fundamental idea of ray-casting is to trace rays for the synthesis of images. For each pixel of the image plane a ray is shot from the eye of the viewer through the respective pixel into the object space (sometimes also in the reverse direction) and traced for determining the color of the pixel. Thus, ray-casting is an *image space* method where for each pixel the contribution from the objects in the object space is determined. In contrast, the rasterization approach for rendering triangle meshes is an *object space* method where for each triangle the contribution to the pixels is determined. Ray-casting does not a priori anticipate the way how the color of a pixel is computed while tracing the ray. In particular, ray-casting is not limited to volume visualization but can also be used to render triangle meshes. In this case, the intersection point of the ray with the nearest triangle is computed.

The concept of ray-casting has been known for decades in computer graphics [Appel, 1968] and was extended to ray-tracing by Whitted [Whitted, 1979]. In contrast to ray-casting, ray-tracing considers not only a single ray per pixel but recursively generates rays for computing reflections, refractions and shadows once the primary ray hits an object. Ray-tracing is known for its excellent image quality but also for its tremendous demand for computing power making it unsuitable for interactive applications. In fact, ray-tracing has a long history in the offline rendering community including Pixar's RenderMan [Jensen and Christensen, 2007], which is used for producing the popular animated feature films.

Even though ray-casting has been suggested for volume visualization [Levoy, 1988] a long time ago, it is the GeForce 8 series GPUs [NVIDIA, 2006] that brought a breakthrough. The support for executing conditional loops with an arbitrary number of instructions and the tremendous processing power makes it fast and easy to implement interactive volume ray-casting on the GPU. In the following, the necessary steps are explained. An overview of the basic algorithm is presented in Algorithm 1. Additional information can be found in these documents [Hadwiger et al., 2009; Engel et al., 2006; Scharsach, 2005].

Algorithm 1 Basic algorithm of GPU volume ray-casting

{——————————— Setup ———————————}
$V \leftarrow$ volumetric dataset
$\nabla V \leftarrow$ compute gradient as normal for V using central differences
$B \leftarrow$ compute bounding geometry of V
$T_V \leftarrow \nabla V, V$ {store ∇V and V to 3D texture}
$T_{TF} \leftarrow$ transfer function {store transfer function to 1D texture}
for all frames **do**
  {——————————— Generate rays ———————————}
  Set texture coordinates for B
  Enable rendering of back faces only
  $T_{back} \leftarrow$ render B {store exit points of the rays to 2D texture}
  Enable rendering of front faces only
  $T_{front} \leftarrow$ render B {store entry points or the rays to 2D textures}
  {——————————— Render ———————————}
  Enable GPU program for rendering
  Render B
  {——————————— GPU program ———————————}
  for all pixels $\in B$ at (x, y) **do**
    $v_{entry} \leftarrow T_{entry}(x, y)$ {fetch entry point}
    $v_{exit} \leftarrow T_{exit}(x, y)$ {fetch exit point}
    $p(t) \leftarrow v_{entry} + t * |(v_{exit} - v_{entry})|$ {define the ray}
    $r \leftarrow 0$ {initialize result for pixel}
    $t \leftarrow 0$ {initialize loop}
    {——————————— Sample along the ray ———————————}
    while $t \leq 1$ **do**
      $v_{rgb,\alpha} \leftarrow T_V(p(t))$ {fetch normal v_{rgb} and scalar value v_α}
      $c_{rgb,\alpha} \leftarrow T_{TF}(v_\alpha)$ {fetch color c_{rgb} and opacity c_α}
      $r_{sample} \leftarrow$ evaluate illumination model using c_{rgb}, c_α, and v_{rgb}
      $r \leftarrow r+$ contribution from r_{sample} {according volume rendering integral}
      $t \leftarrow t+$ sample interval
    end while
    framebuffer$(x, y) \leftarrow r$ {store result to frame buffer}
  end for
end for

2.4.1 Setup

Before rendering a volumetric dataset V (see Section 2.1) on the GPU some preparations have to be made. First, the normal for each voxel of V is required for evaluating the illumination model during rendering (assuming a Blinn-Phong [Phong, 1973; Blinn, 1977] like illumination model) if they are not already provided.

Computing the normals in a preprocessing step is advantageous since it has to be done only once per dataset and it avoids computing them during the actual rendering pass. Commonly the normals are approximated by the gradient ∇V, which itself can be approximated by central differences [Engel et al., 2006]:

$$\nabla V_{i,j,k} = \begin{pmatrix} \frac{\partial V}{\partial x} \\ \frac{\partial V}{\partial y} \\ \frac{\partial V}{\partial z} \end{pmatrix} \approx \begin{pmatrix} V_{i+1,j,k} - V_{i-1,j,k} \\ V_{i,j+1,k} - V_{i,j-1,k} \\ V_{i,j,k+1} - V_{i,j,k-1} \end{pmatrix} \tag{2.30}$$

Second, the volumetric dataset V including the normals needs to be transferred to the graphics hardware into the GPU memory for fast access. If V including normals fits entirely into the GPU memory, it just can be uploaded into a 3D texture [Shreiner et al., 2007] where the color channels are used for the normals and the α channel for the voxel values. Values and normals can then be obtained by a simple texture lookup with texture coordinates $(s, t, r) \in [0, 1] \times [0, 1] \times [0, 1]$. Depending on the parameters set for the 3D texture, the values and normals will be automatically interpolated with a trilinear filter. If V with normals does not fit into the GPU memory, bricking can be employed [La Mar et al., 1999], which is a standard technique but not further discussed in this thesis.

Third, the transfer function used during rendering needs to be loaded into an additional, auxiliary 1D texture (assuming a one-dimensional transfer function). The setup is now finished except that some parameters such as the position of the light source have to be set and the actual rendering can take place.

2.4.2 Rendering

In a first pass the rays need to be generated with the help of the entry and exit points of the rays into and out of the volumetric dataset respectively, which are stored in auxiliary 2D textures (Figure 2.7). For this, a bounding geometry of the volumetric dataset is required. In the simplest case this is only a box enclosing the volumetric dataset but it can also be a more sophisticated geometry like an octree [Samet, 1990] allowing for cutting away empty regions (and thus enhancing the performance). The bounding geometry is rendered twice directly into the respective auxiliary texture where the first time only the front faces are rendered and the second time only the back faces. The key point is that the entry and exit positions defining the rays are encoded into the color channels of the bounding geometry. This can be achieved by setting the color of the lower left front corner of the bounding geometry to $(0, 0, 0)$ and the color of the upper right back corner to $(1, 1, 1)$ and thus corresponding to the spatial position on the bounding geometry. While rendering the bounding geometry into the auxiliary textures, the colors will be interpolated automatically resulting in the correct entry and exit points. Given the position of a pixel and given the entry point v_{entry} and exit point v_{exit} for that

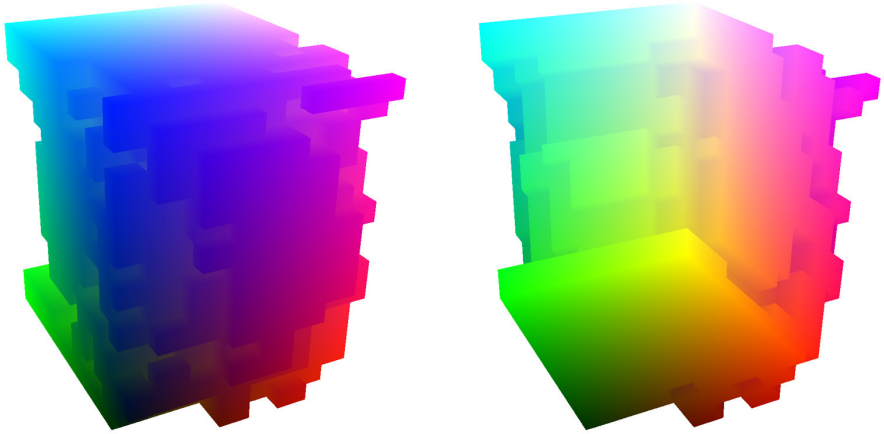


Figure 2.7: *The bounding geometry of the Visible Human dataset. On the left the front faces are rendered and on the right the back faces. The color values are interpolated according to the spatial position on the bounding geometry. Consequently the color values in the left image contain the entry points and the color values in the right image the exit points for the rays through the respective pixels. Dataset courtesy of the National Library of Medicine, National Institutes of Health, USA.*

pixel from the auxiliary textures, the direction vector is obtained by subtracting v_{entry} from v_{exit} and the ray in its parametric form is defined as:

$$p(t) = v_{entry} + t|(v_{exit} - v_{entry})| \tag{2.31}$$

Now everything is prepared for the actual rendering pass. A GPU program, for example in terms of a GLSL shader (OpenGL Shading Language [Rost, 2006], a programming language for writing GPU programs), is enabled and the front faces of the bounding geometry are rendered once again. For each pixel of these front faces, the GPU program is executed. It basically fetches the entry and exit point from the respective auxiliary texture and computes the ray. Then, samples from the 3D texture are taken along the ray according to a predefined interval (see also Figure 2.1). Each sample value consisting of a scalar value and associated normal is classified by looking up the 1D texture containing the transfer function with the scalar value. Next, the illumination model is evaluated using the color and opacity returned from the transfer function lookup, the normal, and a predefined light source. Finally, the contribution of the sample is added to the contributions of the previous samples according to Equation 2.26 and after the last sample, the final color is written into the frame buffer. Once all pixels are finished the final image can be presented to the user.

Many optimizations and variations have been proposed to this very basic algorithm for implementing volume ray-casting on today's GPUs. For example, the performance can be enhanced by stopping sampling when the intermediate result becomes opaque or by only sparsely sampling unimportant areas [Ljung, 2006]. In Chapters 3 and 4 a GPU volume ray-caster is used as a basis for implementing the visibility-difference entropy transfer function generation and the extinction-based illumination and shading.

2.5 Splatting

In contrast to ray-casting splatting is an object space method. The basic idea is to determine the contribution of each voxel of the volumetric dataset to the pixels in the image plane. For this, the voxels are first sorted according to their position perpendicular to the image plane and the transfer function is applied to each voxel, including application of an appropriate illumination model. Then, each voxel is projected onto the image plane one after another in the sorted order adding its contribution to the affected pixels. In particular, the projection of a voxel is performed by weighting the 2D footprint of a 3D interpolation kernel by the color and opacity of the voxel and rasterizing the footprint in the image plane. The process of rasterizing the footprint is called *splatting* (see Figure 2.8). During splatting the contribution from the footprint is blended with existing contributions from preceding footprints in the image plane leading to the final color.

Splatting was initially proposed by Westover [Westover, 1989] and became a popular method for volume visualization due to its performance compared to other methods at that time. Additionally, it is possible to use 2D texture mapping hardware for the splatting operations to further enhance the performance [Crawfis and Max, 1993]. Texture mapping hardware was available long before the first programmable GPUs. On the other hand, the quality of the images generated by the original algorithm is not convincing. The images suffer from an inherent blurriness resulting from coarse evaluation of the volume rendering integral and from the application of the transfer function early in the process. Many approaches for solving the problems have been presented [Westover, 1990; Mueller and Crawfis, 1998; Mueller et al., 1999; Huang et al., 2000; Zwicker et al., 2001] and will be discussed in the following subsections.

In recent years splatting on the GPU has been suggested [Neophytou and Mueller, 2005; Neophytou et al., 2006; Grau and Tost, 2007; Chen et al., 2004]. Although splatting on the GPU definitely boosts the performance, the limitation is still defined by the rasterization bound of the graphics hardware. Everything that helps deferring this bound like reducing the number of individual splatting operations and the size of the splats improves the performance. However, for achieving

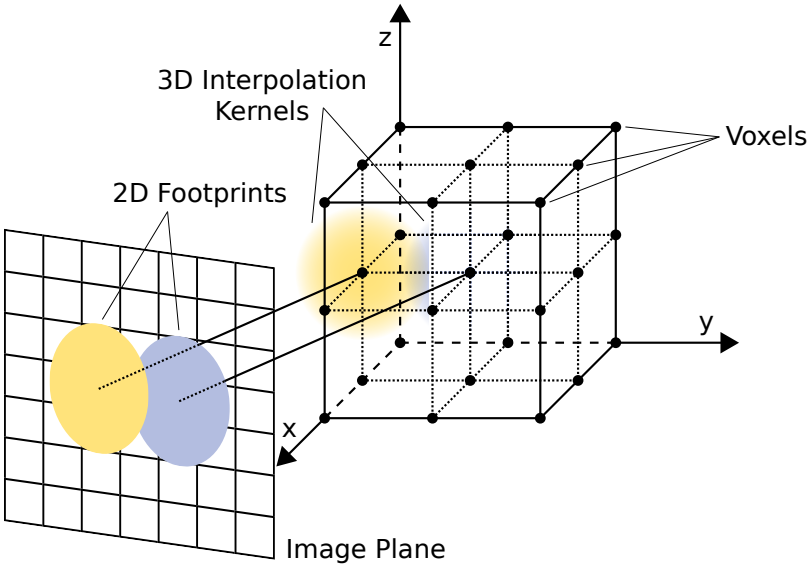


Figure 2.8: *The basic concept of splatting. 3D interpolation kernels are applied to the voxels and 2D footprints of these 3D interpolation kernels are projected onto the image plane.*

a high image quality it is required to splat multiple slabs of a kernel as will be shown in the following subsections. In Chapter 5 we present an approach that limits the number of required splatting operations to exactly one per voxel. Using a correction term to compensate for not splatting multiple kernel slabs makes it possible to enhance the performance while not compromising the image quality at the same time.

2.5.1 Interpolation

An important part of splatting is the way how the interpolation or reconstruction of the volumetric data is concluded. A 3D interpolation kernel, for example the Gaussian from Equation 2.13, is applied to each voxel. The value at an arbitrary position (x, y, z) can then be obtained by summing up the contributions from all interpolation kernels at that position:

$$\sum_{i,j,k \in D} V_{i,j,k} \cdot s(i - x, j - y, k - z) \tag{2.32}$$

where V is the volumetric dataset, $D \subset \mathbb{Z}$ is the definition range of the volumetric dataset, and s is the interpolation kernel. Hence, the individual contribution of the

voxel at position (i, j, k) to a sample at an arbitrary position (x, y, z) is:

$$V_{i,j,k} \cdot s(i - x, j - y, k - z). \tag{2.33}$$

The volumetric dataset can be reconstructed by iterating through all voxels and adding their contributions to all samples at affected positions.

Considering orthogonal projection where the view ray through each pixel in the image plane is perpendicular to the image plane and taking into account that the discretized volume rendering integral from Equation 2.26 is basically a sum, the following approximation can be made. Instead of determining the contribution of the voxels in the 3D object space, the contribution is determined directly in the 2D image space using a 2D footprint of the 3D interpolation kernel. This 2D footprint (FP) is generated by integrating the interpolation kernel along the view ray (in this example z):

$$FP(x, y) = \int_{-\infty}^{\infty} s(x, y, z)dz. \tag{2.34}$$

The contribution of a voxel at (i, j, k) to a sample at position (x, y) in the image plane is then:

$$V_{i,j,k} \cdot FP(i - x, j - y). \tag{2.35}$$

The obvious problem with this approximation is that the visibility is not properly determined within the extent of the interpolation kernel in the direction of the view ray. The integration from the volume rendering integral is substituted by the pre-integrated footprint without continuous attenuation as required. Only when splatting the footprint onto the image plane the attenuation is evaluated for the entire extent of the interpolation kernel along the view ray by blending with preceding contributions. This leads to the blurriness and artifacts but has been solved by using sheet-buffers as discussed below.

2.5.2 Axis-aligned and View-aligned Sheet-buffers

The problem of the improper visibility determination was discovered soon and addressed by introducing the concept of sheet-buffers [Westover, 1990; Mueller and Crawfis, 1998]. A number of parallel sheets serving as intermediate planes is used to subdivide the volumetric dataset. The projection of the voxels is not performed directly onto the image plane anymore but onto these sheets, which are then composited for the synthesis of the final image. Intentionally, the distance between two consecutive sheets is much smaller than the extent of the 3D interpolation kernel, requiring the kernel to be cut into slabs (see Figure 2.9). These slabs are integrated along the direction perpendicular to the sheets (similar to Equation 2.34) resulting in an number of slab footprints. For projecting a

voxel, the 3D interpolation kernel is applied to the voxel and the sheets affected by the 3D interpolation kernel are determined. The actual projection is performed by splatting the appropriate slab footprints of the 3D interpolation kernel to the affected sheets (see Figure 2.10).

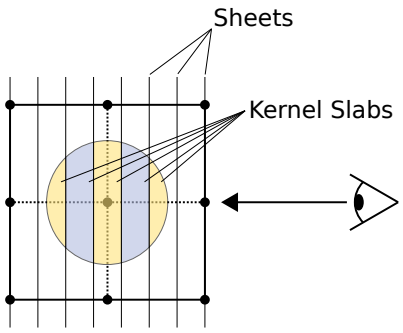


Figure 2.9: *The volumetric dataset is subdivided using sheets. The 3D interpolation kernel is cut into five slabs by the sheets and each slab contributes to the respective sheet.*

Splitting up the 3D interpolation kernel into several slabs and determining the visibility independently for each slab on the respective sheet approximates the volume rendering integral much closer than the original method. It leads to a superior image quality diminishing blurring and artifacts. Further, a cheap bucket sort ($O(n)$, see [Heineman et al., 2008]) where the buckets correspond to the sheets is sufficient since the splatting order within a sheet is not important. In contrast, the original method requires a complete sort of the voxels as a preprocessing step (typically $O(n \log n)$). However, splitting the 3D kernel up into several slabs and splatting the footprint of each slab onto the appropriate sheet multiplies the number of splatting operations. This increase in splatting operations heavily affects the performance since the rasterization bound is the limiting factor for the performance of splatting. Another thing to consider is that the voxels may lie anywhere between two sheets defining an arbitrary offset for cutting the 3D interpolation kernel into slabs. Thus, an entire set of slab footprints has to be kept for any offset between two sheets.

Westover [Westover, 1990] suggested axis-aligned sheets where the axis perpendicular to the sheets is the one with the smallest angle to the view direction. Only if rotation goes over a 45°angle the orientation of the sheets changes and only in this case a resort of the voxels is required. Contrariwise axis-aligned sheets suffer from visible popping artifacts when rotating over a 45°angle and the orientation of the sheets changes. This phenomenon is attributed to the still coarse approximation of the volume rendering integral producing different results for different sheet orientations. To get rid of the popping artifacts, view-aligned sheets [Mueller and Crawfis, 1998] have been proposed. View-aligned sheets are

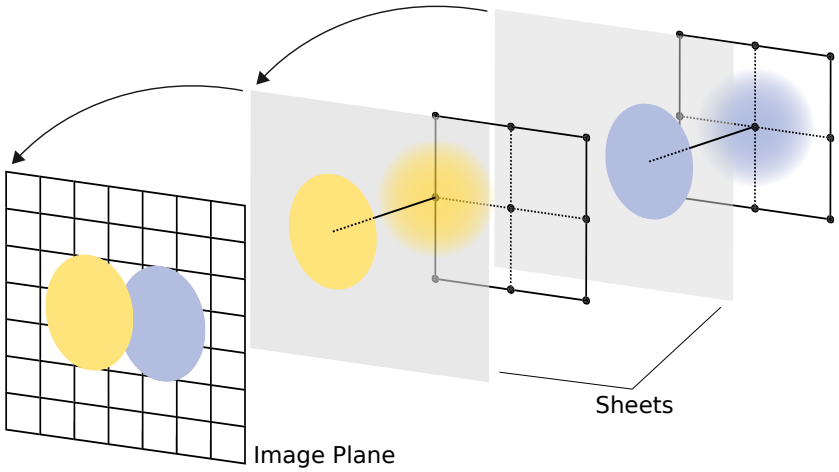


Figure 2.10: *Volume splatting using sheets. The splatting operations are performed onto the sheets before compositing them for the final image.*

- as the name already suggests - always perpendicular to the view direction requiring a resort for each change of the view direction. The advantage of view-aligned sheets consists of the absence of popping artifacts when rotating the dataset over a 45°angle. There is never a big change of the orientation of the sheets relative to the dataset orientation for a small rotation as with axis-aligned sheets.

2.5.3 Post-Classification

Applying the transfer function in the context of splatting is often called classification. The original splatting algorithm puts this classification right in the beginning before applying the interpolation kernel, which is known as *pre-classification*. With pre-classification the interpolation kernel does not interpolate the actual scalar values from the volumetric dataset but colors and opacities, contributing to the blurring and color bleeding artifacts of the original method.

Moving the classification to after interpolation of the scalar values from the volumetric dataset can enhance the image quality, and in conjunction with sheet splatting get rid of blurring and color bleeding artifacts entirely [Mueller et al., 1999]. Assuming sheet splatting, the voxels are bucket sorted to the sheets in a preprocessing step. For each sheet the voxels from the respective bucket are then splatted onto that sheet by applying the appropriate footprint of the interpolation kernel slab directly to the scalar values of the voxels. Once a sheet is completed, it contains the interpolated scalar values. Only then the transfer function is applied and the illumination model is evaluated for each pixel of the sheet before

finally compositing it with the other sheets. Moving the classification to after interpolation is called *post-classification*. Using view-aligned splatting together with post-classification delivers the best image quality depending on the interpolation kernel used. At the same time it is very expensive due to the sorting required on each change of the view direction, the additional splatting operations from the sheets, and the fact that classification is required for every pixel on every sheet instead of just once per voxel.

2.5.4 Implementation on the GPU

Algorithm 2 demonstrates a hybrid CPU/GPU approach for volume splatting. In a preprocessing step the normals for the voxels of the volumetric dataset are computed if they are not already provided (similar to GPU ray-casting in Section 2.4). Next, the voxels are distributed to the sheets using bucket sort where each voxel appears in all buckets of the respective sheets it affects. The buckets are then transferred to the GPU using a vertex buffer object (VBO) and subsequently all sheets are processed one after another front-to-back or back-to-front.

Point sprites [Shreiner et al., 2007] and a GPU program are used to splat the voxels from the respective buckets to the sheets. This GPU program basically picks the right footprint from a footprint map stored on the GPU as 2D texture and determines and sets the right size for the footprints. While rasterizing the footprints their contribution is added to existing contributions from other footprints on the particular sheet. Once a sheet is finished it can be composited with the previous sheets using another GPU program. This GPU program fetches the color and opacity from the transfer function stored in a 1D texture on the GPU for each pixel of the sheet and evaluates the illumination model. Finally, the contribution of each pixel is composited with the previous sheets according to the volume rendering integral.

With some effort it is possible to implement the sorting on the GPU as well [Grau and Tost, 2007] to get the desired GPU only approach providing additional performance gains. Further, the normals approximated by gradients in terms of central differences can be computed on the fly during compositing. For this, it is required to keep at least three sheets at the same time on the GPU such that for a particular pixel the central differences can be computed by looking up these three sheets. Not providing pre-computed normals has the advantage that the color channels used for the normals can be exploited otherwise. Neophytou and Mueller [Neophytou and Mueller, 2005] use these color channels for a method to splat four footprints of kernel slabs at the same time and thus deferring the rasterization bound of the graphics hardware. They exploit a special property of spherically symmetric Gaussian kernels (see Equation 2.13) where footprints of kernel slabs can be derived from each other by simply multiplying them with a factor. Hence,

the color channels are used to store the factors for the kernel slabs of four consecutive sheets and consequently the footprints of four consecutive slabs can be splatted at the same time. On the other side, computing the normals on the fly comes at the price of an increased number of texture lookups, which may lead to another bottleneck.

Algorithm 2 Algorithm for a CPU/GPU hybrid approach for volume splatting

```
1: {———————————————— Setup ————————————————}
2: V ← volumetric dataset
3: ∇V ← compute gradient as normal for V using central differences
4: S_{1..n} ← setup n sheets either axis- or view-aligned
5: B_{1..n} ← ∇V, V {bucket sort voxels to sheets 1..n and store on the GPU}
6: T_footprints ← footprint map {store all footprints to 2D texture}
7: T_TF ← transfer function {store transfer function to 1D texture}
8: for all frames do
9:    for i = 1 → n do
10:      {————————————————- Process sheet ————————————————}
11:      Bind S_i as render target
12:      Enable GPU program for splatting
13:      for all voxels v ∈ B_i do
14:         {———————————————— Splat voxels ————————————————-}
15:         {using point sprites}
16:         Determine required footprint for v
17:         Setup texture coordinates for T_footprints accordingly
18:         Determine and set size of the footprint
19:         Rasterize footprint
20:      end for
21:      for all pixels p ∈ S_i at (x, y) do
22:         {————————————————- Composite ————————————————}
23:         {normal p_rgb and scalar value p_α}
24:         c_{rgb,α} ← T_TF(p_α) {fetch color c_rgb and opacity c_α}
25:         r ← evaluate illumination model using c_rgb, c_α, and p_rgb
26:         {according volume rendering integral}
27:         framebuffer(x,y)←framebuffer(x, y)+ contribution from r
28:      end for
29:   end for
30: end for
```

2.6 IVS

IVS is the volume visualization system developed by the author of this thesis at the Visualization and Multimedia Lab of the University of Zurich. It is the ab-

breviation for **I**nteractive **V**olume **S**platter because it was originally designed as a volume splatting system but soon it was extended to incorporate 3D texture slicing and GPU ray-casting as well. The architecture of IVS is modularly separated into volumetric dataset handling, transfer function handling, renderers, evaluation features, and a GUI. The entire framework is written in C++ using OpenGL[1], boost[2], vmmlib[3], libpng[4] and Qt[5] as its only dependencies. It runs on any platform given that the dependencies are available on the desired platform and that the version of OpenGL matches at least 2.1. IVS is the basis for the implementation of all contributions in this thesis. A short discussion of the features and limitations of IVS is provided as follows:

- **Dataset Handling**

  IVS supports the import of volumetric datasets in the raw format (just the values) with normals provided optionally. The scalar data can either be 8 or 16 bit wide per value. Internally a copy of the dataset is kept in an octree data structure where the depth of the octree is dynamically adjusted depending on the size of the dataset. Normals are computed in a preprocessing step if they are not already provided with the dataset and zero values are omitted if not explicitly specified otherwise. Keeping the dataset in an octree data structure allows for executing many operations on the dataset in parallel and in addition, a bounding geometry for GPU ray-casting can be created very easily. As of now IVS only supports volumetric datasets defined on a rectilinear grid.

- **Transfer Function Handling**

  IVS supports one-dimensional transfer functions for color and opacity as well as for material properties either for the scalar values of the volumetric dataset or for the gradient of the scalar values. The transfer functions can be freely designed by drawing a curve in a GUI or by choosing either a ramp, a Gaussian, a double Gaussian or a ramp with a Gaussian as basis function and manipulating its parameters (see Figure 2.11). Once created, transfer functions can be exported to a file and later the file can be imported again. The functionality for automatic, visibility-difference entropy transfer function generation is explained in Chapter 3. Currently, IVS does not support multi-dimensional transfer functions.

[1] www.opengl.org
[2] www.boost.org
[3] vmmlib.sourceforge.net
[4] www.libpng.org
[5] qt.nokia.com

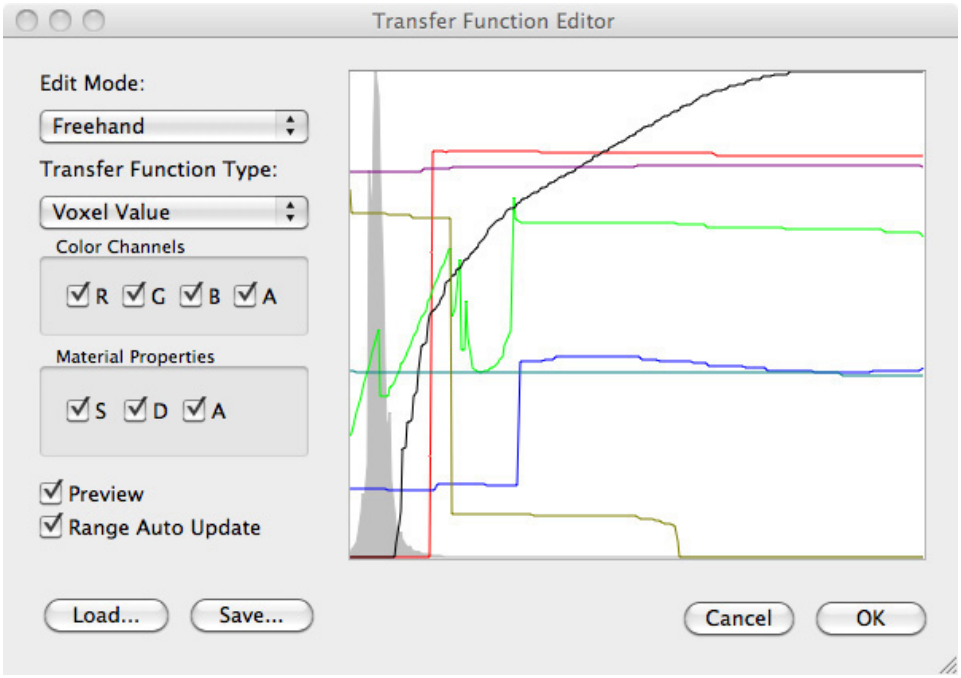


Figure 2.11: *The transfer function editor of IVS.*

- **Renderers**

  IVS incorporates renderers for GPU ray-casting as described in Section 2.4, for 3D texture slicing [Engel et al., 2001] and for splatting on the GPU as described in Section 2.5. GPU ray-casting is extended for extinction-based illumination and shading in Chapter 4 and splatting is evolved to layered splatting in Chapter 5. Despite this progress it is still possible to choose the original renderers, mainly for comparison purposes. The renderers can be adjusted by a plethora of parameters presented to the interested user. As of today, neither time varying datasets nor rendering of multiple datasets loaded at the same time is supported.

- **Evaluation Features**

  Dedicated features have been implemented for the evaluation of enhancements. This starts from measuring the timings for rendering individual images, for default rotations around axes and for entire sessions. Further, simple screenshots can be taken or movies can be captured in two different ways. The first way is to capture movies of a default rotation with a fixed angular velocity independent of the time required to render. This always leads to very smooth animations well suited for judging the quality of an

enhancement. The second way is to capture a user defined movie where screenshots are taken at a fixed interval and quickly stored in the main memory degrading the performance as few as possible. These movies are well suited for judging the speed, the interactivity, and the user experience of an enhancement.

- **GUI**

  The GUI of IVS is built using the platform independent framework Qt. This enables IVS to run on many platforms including the major PC operating systems. Nevertheless, the logic of the volume visualization system, in particular the renderers, has been encapsulated as far as possible not relying on the GUI framework but on OpenGL only. This allows for an easy exchange of the GUI framework should it ever be required. At current state, the GUI is clearly targeted to research, not containing the consistency and ease of use expected from a production system. Figure 2.12 shows the main window of IVS.

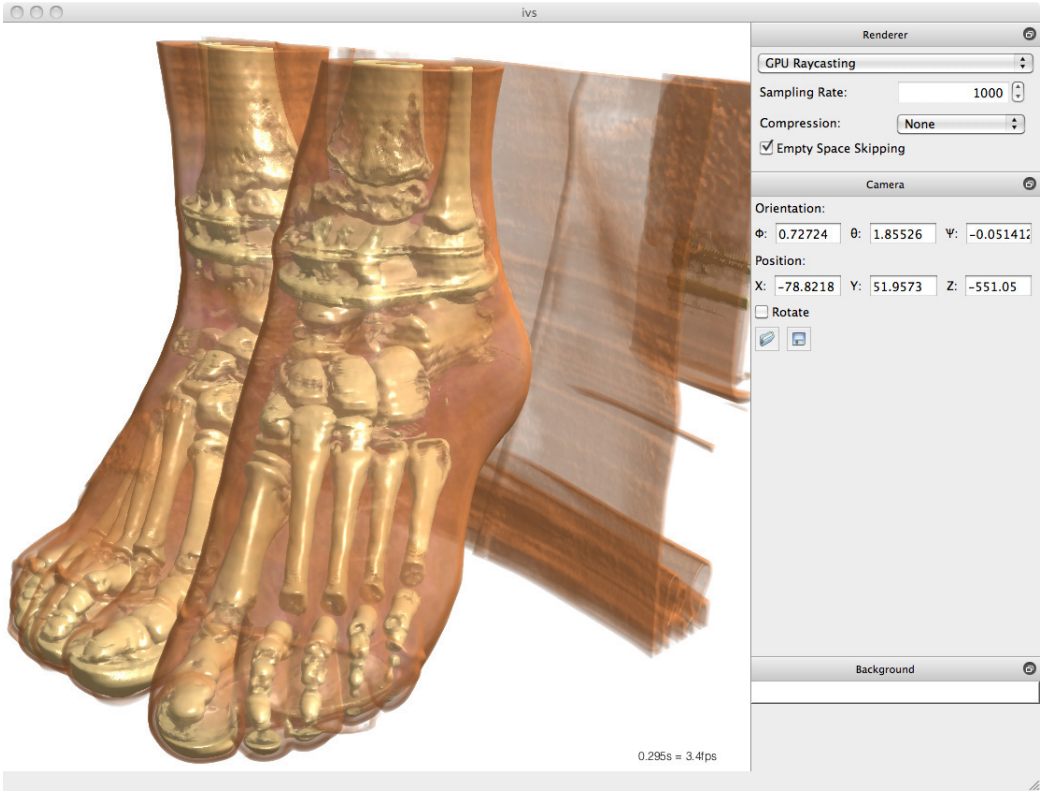


Figure 2.12: *The main window of IVS showing the feet dataset visualized using GPU ray-casting. Dataset courtesy of OsiriX.*

CHAPTER 3

VISIBILITY-DIFFERENCE ENTROPY TRANSFER FUNCTION GENERATION

3.1 Automatic Transfer Function Generation

A very important part of volume visualization is the transfer function, specifying the transition from the raw scalar data of the volumetric dataset to the color and opacity space (see Section 2.2). Consequently the transfer function constitutes the appearance of the final image by defining what parts and what details of the volumetric dataset are revealed, what parts are highlighted by a salient color catching the attention of the viewer, and what parts are displayed semi-transparent supporting the overall perception. Applying the right transfer function is crucial for domain experts to tease out of the volumetric dataset the information they are looking for. However, the manual design of good transfer functions is a very time consuming, unintuitive, and difficult task demanding for automatic or semi-automatic solutions. The vast majority of such solutions are based on the analysis of the volumetric dataset in the object space, focusing on and exploiting specific features of the dataset. Often targeted to specific domains, the underlying algorithms always generate the same kind of transfer functions accentuating the properties they have been designed for. This is reasonable as long as these algorithms are used in an expert way knowing the basic shape of the dataset in advance and

being interested in exactly the features revealed by the particular technique. Contrariwise, there are far less general purpose approaches allowing for insight into an entirely unknown dataset. In this thesis we propose such a general purpose approach based on information theory. Unlike the majority of existing methods, our method is computed in the image space generating not only a single transfer function but a set of transfer functions. The resulting images have the highest information content while being as much different from each other as possible when applying the individual transfer functions. With our approach it is possible to gain insight into each facet of a completely unknown dataset without focusing on specific properties and features.

3.1.1 Object-Space Methods

Object-space methods analyze and use the properties and features of the volumetric dataset itself. Among the important properties are the first and second order derivatives revealing surfaces at positions where they are non-zero. Kindlmann et al. [Kindlmann and Durkin, 1998] use the data value, the first and second order derivatives along the gradient direction and capture their relationship and thus the boundary information in a histogram volume. The user can then control which portions of the boundary should be rendered opaque.

Apart from the derivatives many approaches use topological attributes of the dataset. Bajaj et al. [Bajaj et al., 1997] propose the contour spectrum consisting of scalar data and contour attributes computed over a range. This includes parameters such as surface area, volume, and gradient integral etc., which are presented to the user as signature graphs to help select the right visualization parameters. More recent work [Zhou and Takatsuka, 2009] also uses topological attributes derived from the contour tree of the dataset. The opacities are distributed over the branches of the contour tree by a residue flow model based on Darcy's Law. Further, it is shown that the topological attributes can be used to generate harmonic color transfer functions as well. Hyper Reeb graphs [Fujishiro et al., 1999; Weber and Scheuermann, 2004] are another way for generating transfer functions based on topological attributes. The basic idea is to identify and highlight critical iso-surfaces from the hyper Reeb graph and keep the change in hue as well as the opacity constant except for the critical iso-surfaces.

Other interesting ideas include the generation of multi-dimensional transfer functions by presenting to the user slices of the dataset with the option to paint the regions of interest [Tzeng et al., 2003]. A neural network is then used to generate the multi-dimensional transfer function. General regression neural networks are also suitable for adaptive transfer function design [Zhang and Sun, 2003]. Further, neural networks are popular for generating segmentations in medical imaging [Duch and Jankowski, 1997; Hall et al., 1992]. Focused on the classification

of tissue [Sato et al., 2000], 3D filters based on the gradient vector and the Hessian matrix of the volume intensity function can be employed to generate multi-dimensional transfer functions. To design these filters not only intensity values are taken into account but also the local neighborhood to identify line-, sheet-, or blob-like structures that are typical for blood vessels, bone cortices, etc. Another method for tissue detection in unknown datasets is based on partial range histograms [Lundstrom et al., 2006]. Tissue is detected by a peak pattern in ranges of local intensity histograms. The visualization of boundaries is the goal of an approach based on low-high histograms [Šereda et al., 2006]. They allow for an easy and more robust selection of boundaries than scalar value and gradient magnitude.

3.1.2 Image-Space Methods

In contrast to object space methods, image space methods do not analyze the properties and features of the volumetric dataset. They rely on the generated images and often adjust the parameters for generating the transfer function in a feedback loop. This also includes semi-automatic methods where the adjustment of the parameters is directed by the user after inspecting the generated images. Early work includes a model that defines a transfer function as a sequence of 3D image processing procedures [Fang et al., 1998]. In particular, the transfer functions are represented as a sequence of intensity mappings either as look-up tables or neighborhood functions.

The design of transfer functions can also be treated as a parameter optimization problem [He et al., 1996]. Based on stochastic algorithms an initial set of transfer functions is generated, which is then evaluated by the user. The search is repeated until a satisfying result is obtained. A genetic algorithm is used to search for the global optimum where the required fitness can be influenced interactively by the user or assigned automatically using entropy, variance, or edge energy. Similarly a particle swarm can be used to solve the optimization problem [Li et al., 2009]. Design galleries [Marks et al., 1997] are a method if neither interactive evolution is suitable nor the output quality can be quantified mathematically, thus making automatic optimization impossible. By varying a vector of input parameters a set of perceptually varying images is generated. The dispersion algorithm guarantees a set of input parameters that maps to a well-distributed set of output images. A recent approach [Correa and Ma, 2009; Correa and Ma, 2011] introduces visibility-driven transfer functions where visibility histograms representing the visibility for a sample from a given viewpoint are defined. These visibility histograms are used as a feedback mechanism for manual and automatic generation of transfer functions.

A different way of making the design of transfer functions easier is to provide a good interface [König and Gröller, 2001; Rezk Salama et al., 2006; Wang and

Mueller, 2008] or widgets [Kniss et al., 2002a] to the user.

3.1.3 Visual Quality and Saliency

The evaluation of transfer functions in terms of quality of the resulting images is not directly related to the transfer function generation but nevertheless important for automatic solutions using a feedback loop. The literature is rather sparse when it comes to define a metric for quantifying the quality of transfer functions or images generated by particular transfer functions. However, the problem of saliency or finding salient viewpoints is closely related and requires a similar metric. One way of choosing a salient viewpoint is based on obscurance [Ruiz et al., 2008]. Obscurance represents the occlusion information associated with the voxels and the best viewpoint is computed from the variation of obscurance of the visible voxels.

An alternative to obscurance is the viewpoint entropy [Vázquez et al., 2004], which is based on the projected area of faces related to the total area of faces. Replacing the viewpoint entropy by a linear combination of the viewpoint entropy, luminance, and chrominance is another variation [Nakagawa et al., 2006]. A further evolution of the viewpoint entropy leads to the viewpoint mutual information, which is a channel between a set of viewpoints and the viewpoint entropy [Feixas et al., 2009]. Coming from triangle meshes, the viewpoint mutual information has the advantage that it is invariant to mesh subdivision and that it will converge to an upper bound. Instead of the viewpoint entropy, the entropy of the intensity image of the visible boundary structures relative to the viewpoint can be taken [Tao et al., 2009]. Basically, the boundary structures describe the shape of the objects from the volumetric dataset and the variance between the shape of the objects and the original objects describes the details. This measurement of the details can also be used for viewpoint selection.

Semantic driven approaches for viewpoint selection rely on view-dependent shape properties [Mortara and Spagnuoloa, 2009]. The viewpoint is selected such that the visibility of meaningful features of the shape are maximized. Another possibility is to choose the viewpoint in a way that the objects based on shape properties can be optimally discriminated from objects in a database [Laga, 2010].

The saliency map [Itti et al., 1998] is focused on scene analysis. First, feature maps are created with center-surround difference for intensity, chromatic opponency red/green, green/red, blue/yellow, and yellow/blue, and local orientation contrast. These feature maps are then combined into a saliency map. The saliency map can be extended to a quality metric [Jänicke and Chen, 2010] where the user defines a relevancy map. The difference between the saliency and the relevancy map indicates structures with little, medium, or high relevance.

3.2 Visibility-Difference Entropy Metric

In this thesis we propose visibility-difference entropy transfer function generation, which is an image space method for generating a set of best transfer functions. As discussed in the previous section, image space methods frequently employ a feedback loop for searching the best transfer function. Often these methods are semi-automatic involving the user to provide feedback according to some criteria for directing the search. Visibility-difference entropy transfer function generation also uses a feedback loop for searching the set of best transfer functions but is fully automatic. The qualifying idea is to introduce a metric derived from information theory taking into account that datasets are interactively and visually explored. Therefore, the information of a transfer function is not measured from a static image but considers the visibility-differences when animating the data. In the following this new visibility-difference entropy metric is defined. For the remainder of the chapter we will abbreviate visibility difference entropy with VDE and call our method *VDE transfer function generation* and the metric *VDE metric*.

3.2.1 Proposition

In recent years interactive volume visualization applications have become mainstream. The goal of VDE transfer function generation is therefore to provide a set of transfer functions that are best in interactive applications. They should reveal as much as possible of the volumetric dataset when rotating, dragging and zooming during an interactive exploration session, and provide a superior structural perception during animation. It is possible that a transfer function satisfying these criteria may not be the best choice if only a single, static viewpoint with a single resulting image is considered. This is fundamentally different from existing approaches focusing on static images.

The basic idea is to use Shannon's entropy [Shannon, 1948] to evaluate the information content as already suggested in previous work (see Section 3.1). But instead of computing the entropy of resulting images, the entropy of differential images from different viewpoints is computed (see Figure 3.1). This means that a transfer function revealing much of the structure of a volumetric dataset when rotating, and thus providing a good structural perception, will be rated high. In contrast a transfer function making the images look the same from any viewpoint not revealing much of the structure will be rated low. Also a transfer function revealing the complex boundary of a volumetric dataset will be rated high since it is likely that this boundary looks different from different viewpoints. The pathological case is a sphere that looks the same from everywhere.

An alternative to the entropy of differential images could be the informational divergence (also Kullback-Leibler divergence) of the images from different view-

points. However, we consciously decided not to take the informational divergence since our method has several advantages. First, the entropy of differential images is symmetric, unlike the asymmetric informational divergence. Symmetry is also prerequisite for a metric in the mathematical sense. Second, by computing differential images not only the statistical distribution of the values is important but also the spatial distribution. This is not the case for the informational divergence where only the statistical distribution is taken into account. To include the spatial distribution helps to better measure the changes when rotating a dataset.

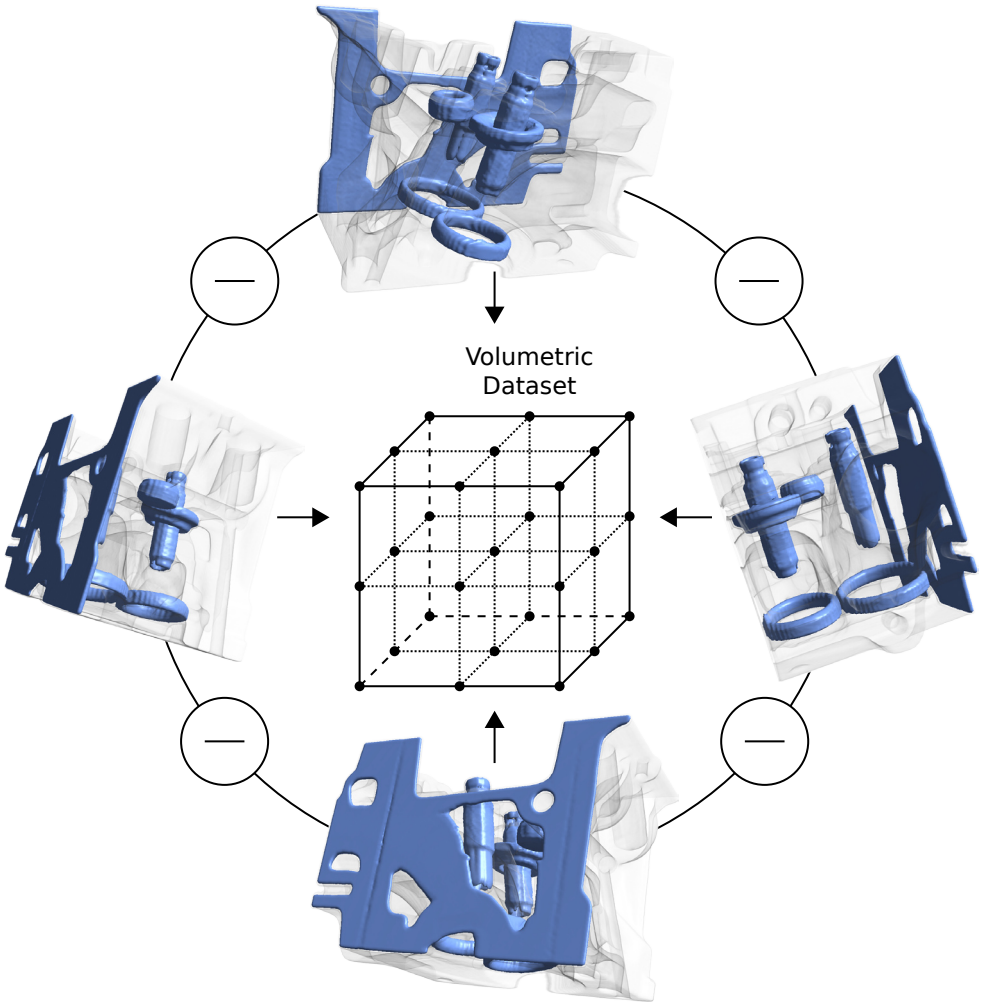


Figure 3.1: *Images for a particular transfer function are rendered from different viewpoints. The VDE metric is then computed from the difference of these images indicated by the minus sign. Dataset courtesy of General Electric.*

Apart from the entropy two additional considerations are made. Because entropy is prone to noise, resulting in high ratings for overly noisy images, a noise

term is applied to the VDE metric. Further, transfer functions that make only small parts of the volumetric dataset visible are not desirable, despite high ratings. Consequently an additional coverage term is applied to the VDE metric. In the following subsections the VDE metric is derived formally starting with the entropy part, adding the noise term, and finally adding the coverage term.

3.2.2 Entropy

The basis for the VDE metric is Shannon's entropy [Shannon, 1948] defined as:

$$H(X) = E(I(X)) \tag{3.1}$$

where X is a discrete random variable, E is the expected value, and I is the information content of X. If X is defined over an alphabet $\{x_1, ..., x_n\}$ and the information content $I(p)$ of the probability p of a character is defined as $-\log p$, the entropy can be written as:

$$H(X) = \sum_{x \in X} p_x I(p_x) = -\sum_{x \in X} p_x \log p_x. \tag{3.2}$$

The maximum entropy is reached if the characters of alphabet X are uniformly distributed:

$$H_{max} = -\sum_{1}^{|X|} \frac{1}{|X|} \log \frac{1}{|X|} \tag{3.3}$$

where $|X|$ is the number of characters in X. The maximum entropy can be used to normalize the entropy.

In the context of VDE transfer function generation the alphabet X is characterized by the color space of a pixel. To avoid empty space having a big impact on the entropy, empty space is not considered and the corresponding value is excluded from the alphabet.

The entropy is computed on the differential images as follows: Given a set of rendered images from different viewpoints R where $r_i \in R$ denotes the i^{th} image, $|r_i|$ the size of the image, and $r_i(m, n)$ corresponds to the value of the pixel at position (m, n), the histogram of a pair of images can be computed as:

$$G_{i,j}(x \in X) = \sum_{(m,n) \in r_i} \begin{cases} 1, & \text{if } |r_i(m,n) - r_j(m,n)| = x \\ 0, & \text{otherwise} \end{cases}. \tag{3.4}$$

The corresponding probabilities are obtained by dividing the histogram by the number of non-zero pixels:

$$P_{G_{i,j}}(x \in X) = G_{i,j}(x) / \sum_{(m,n) \in r_i} \begin{cases} 1, & \text{if } |r_i(m,n) - r_j(m,n)| \neq 0 \\ 0, & \text{otherwise} \end{cases}. \tag{3.5}$$

Consequently, the entropy for a pair of images is:

$$H_{i,j} = -\sum_{x \in X} P_{G_{i,j}}(x) \log P_{G_{i,j}}(x). \tag{3.6}$$

The entropy part of the VDE metric M_H can then be defined as the average of the entropies over all differential images:

$$M_H = \binom{|R|}{2}^{-1} \sum_{i,j \in R \text{ where } j>i} H_{i,j}. \tag{3.7}$$

3.2.3 Noise Term

So far the definition of the VDE metric is defined solely on the entropy of the differential images. A problem with this definition is that the entropy is prone to random, uniform noise leading to high ratings through the uniform distribution of the values. Although volume visualization methods in general do not introduce noise during image synthesis, volumetric datasets obtained by scanning devices may already contain noise. Depending on the transfer function applied, this intrinsic noise may be weakened or it may be amplified. Even if the dataset contains only a layer of noise within a small band, the resulting image may be dominated by noise if the first derivative of the transfer function is high within that band. To compensate for the noise two different measurements of noise are introduced. These noise measurements are computed on the actual images and not on the differential images because the absolute amount of noise is important and not how the noise changes when rotating, dragging or zooming.

The first measure is the well known standard deviation of noise. The basic idea is to subdivide the image into a set K of small patches where r_i^k denotes the k^{th} patch of image r_i and $|r_i^k|$ the size of the patch. For each non-empty patch the standard deviation is computed using the values of the pixels and the overall measure of noise is the average of all patches and images. Given a patch r_i^k the average for that patch is:

$$\text{ø}_i^k = \frac{1}{|r_i^k|} \sum_{(m,n) \in r_i^k} r_i^k(m,n) \tag{3.8}$$

and the standard deviation is:

$$\sigma_i^k = \sqrt{\frac{1}{|r_i^k|} \sum_{(m,n) \in r_i^k} (\text{ø}_i^k - r_i^k(m,n))^2}. \tag{3.9}$$

To get the overall standard deviation as a measure of noise, the average over all patches and all images is computed:

$$\sigma = \frac{1}{|R||K|} \sum_{i \in R} \sum_{k \in K} \sigma_i^k. \tag{3.10}$$

The disadvantage of the standard deviation of noise is that it does not work well if a patch contains a steep color gradient. In this case the standard deviation is quite high even though the patch does not contain any noise at all. Hence another measure is used, which we call pixel deviation. Basically, it measures the average deviation of the pixels within the patches to their direct neighbors. However, taking the absolute value of the difference to the neighbors as deviation is equal to estimating the magnitude of the first order derivative in the direction of the respective neighbor. Therefore, the pixel deviation is also an average of the first order derivatives. The idea behind is that for a color gradient the deviation of a pixel to the direct neighbors is relatively low and hence the derivative is low. On the other hand, if the patch contains a lot of noise a possibly large noise offset is applied randomly to the pixels making the deviation of a pixel to its neighbors high and hence the derivative is high. While the pixel deviation could also be defined on the entire image, the patch overlay is kept to skip empty patches that could falsify the result. Formally, the pixel deviation of a single pixel is:

$$\phi_i^k(m,n) = \frac{1}{4} \sum_{u,v \in \{(1,0),(-1,1),(0,1),(1,1)\}} |r_i^k(m,n) - r_i^k(m+u,n+v)| \tag{3.11}$$

and thus the pixel deviation for a patch is:

$$o_i^k = \frac{1}{|r_i^k|} \sum_{(m,n) \in r_i^k} \phi_i^k(m,n). \tag{3.12}$$

Similar to Equation 3.10 the average over all patches and images is computed as:

$$o = \frac{1}{|R||K|} \sum_{i \in R} \sum_{k \in K} o_i^k. \tag{3.13}$$

Finally, the noise measurements are integrated into the VDE metric in terms of an exponential drop-off window. The window leaves the entropy term unaffected if the noise is not too strong but starts cutting off exponentially once a certain level of noise is reached. Parameters a and b can be used to adjust the drop-off window for level and steepness:

$$M_N = \left(1 - \left(\frac{\sigma}{a_\sigma}\right)^{b_\sigma}\right) \left(1 - \left(\frac{o}{a_o}\right)^{b_o}\right). \tag{3.14}$$

With the noise term integrated, the VDE metric can now be written as:

$$M = M_H M_N. \tag{3.15}$$

3.2.4 Coverage Term

To avoid empty space dominating the VDE metric, empty space was excluded while defining the entropy part. Then again, transfer functions only making a small part of the entire volumetric dataset visible may prevail due to the superior information content of the small part. This is not generally bad but may lead to situations where such transfer functions prevent gaining an overview of the entire dataset. For this reason a coverage term is applied to the VDE metric in order to penalize transfer functions that only make a small part visible.

The coverage term is simply the area covered by non-zero pixels in relation to the entire area:

$$\theta_i = \frac{1}{|r_i|} \sum_{(m,n)\in r_i} \begin{cases} 1, & \text{if } r_i(m,n) \neq 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.16}$$

and the average over all images is:

$$\theta = \frac{1}{|R|} \sum_{i \in R} \theta_i. \tag{3.17}$$

Similar to the noise term, the coverage term is included into the VDE metric as a drop-off window. Parameter b can be used to control the steepness of the drop-off window.

$$M_C = 1 - (1 - \theta)^{b_\theta}. \tag{3.18}$$

The final version of the VDE metric can be written as:

$$M = M_H M_N M_C. \tag{3.19}$$

Images generated by a transfer function can now be evaluated with the VDE metric and thus the transfer function can be evaluated. In a next step, the creation of transfer functions and the search process based on the VDE metric will be explained.

3.3 Basis Transfer Functions

An important part when searching for the transfer functions with the highest ratings is the search space, which is the space of all transfer functions in case of VDE transfer function generation. Without previous knowledge, the search space must be big enough to contain transfer functions with high ratings but it must not be too big since otherwise a search will take too much time. Keeping in mind the characteristics of a one-dimensional transfer function (see Section 2.2) and assuming an 8 bit transfer function defined on a single 8 bit channel, the number of possible

transfer functions is 256^{256}. It is obvious that a search in such a gigantic search space is futile and needs to be restricted.

A possible way of reducing the search space is to use a basis function for the transfer function and then search in the parametric space of the basis function. In the following the basis functions for VDE transfer function generation are presented. Generally, they have four to eight parameters spanning a search space of 256^4 to 256^8 combinations. Even though this is still a huge search space, it is in the reach of randomized search algorithms with acceptable time requirements. Taking the characteristics of the basis functions as transfer functions into account, the search space can further be reduced at a low likelihood of loosing transfer functions with high ratings as shown for the individual basis functions.

- **Gaussian**

  The Gaussian basis function (Figure 3.2(a)) is defined as:

  $$f_G(x) = a \cdot e^{-\left(\frac{x-\mu}{\sigma}\right)^2} + b \tag{3.20}$$

  where a is the amplitude, b is an offset of the basis, μ is the position, and σ is the width of the Gaussian. Consequently the parametric space of the Gaussian consists of these four parameters. However, different amplitudes a do often generate very similar images and raising the base line through parameter b seldom produces images with high ratings. It is therefore reasonable to use only a limited range for parameters a and b or to fix them.

- **Ramp**

  The ramp basis function (Figure 3.2(b)) is defined as:

  $$f_R(x) = \begin{cases} c, & \text{if } x < a \\ c + (b-a)m, & \text{if } x > b \\ c + (x-a)m & \text{otherwise} \end{cases} \tag{3.21}$$

  where a is the beginning of the ramp, b is the end of the ramp, c is the offset from the basis, and m is the gradient of the ramp. Consequently the parametric space of the ramp consists of these four parameters. The parametric space can be slightly reduced by the observation that ramps with $c \neq 0$ for $m > 0$ or $c + (b-a)m \neq 0$ for $m < 0$ seldom produce images with high ratings.

- **Double Gaussian**

  The double Gaussian (Figure 3.2(c)) is basically the combination of two individual Gaussians f_1 and f_2:

  $$f_{DG}(x) = \max(f_{G1}(x), f_{G2}(x)). \tag{3.22}$$

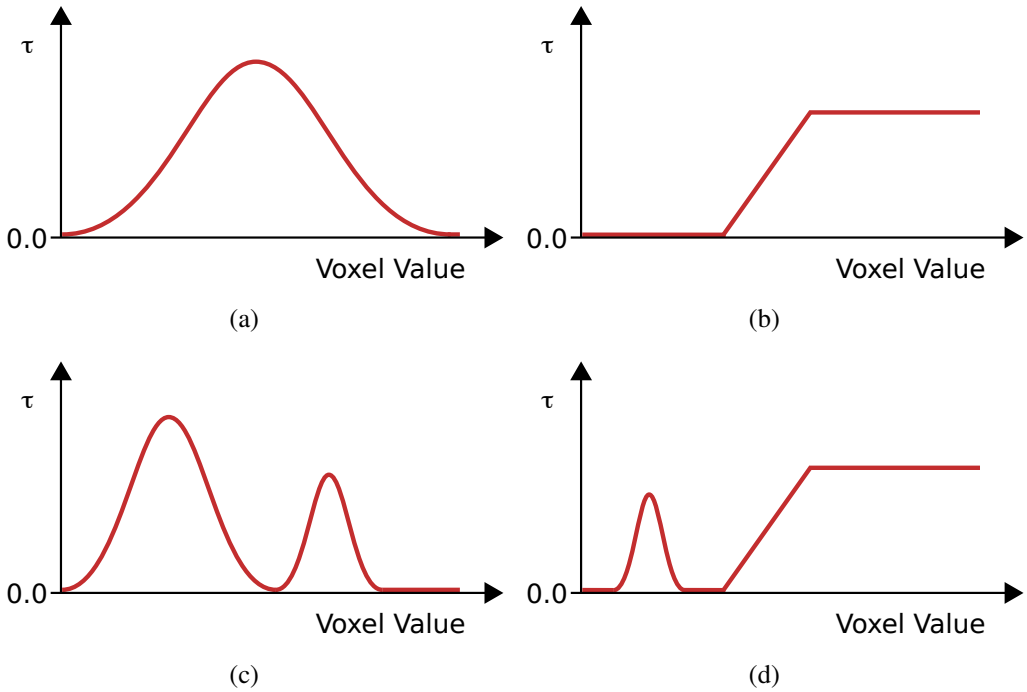


Figure 3.2: *The basis transfer functions: Gaussian (a), ramp (b), double Gaussian (c), ramp Gaussian (d).*

The advantage of the double Gaussian is that two different hot spots in the definition range of the transfer function can be accentuated separately. However, without restrictions, the parametric space of the double Gaussian consists of eight parameters.

- **Ramp Gaussian**

  The last basis function is basically the combination of a ramp with a Gaussian (Figure 3.2(d)):

  $$f_{RG}(x) = \max(f_R(x), f_{G(x)}). \tag{3.23}$$

  Often the ramp is the predominant function with the Gaussian accentuating certain ranges. Without restrictions, the parametric space consists of eight parameters.

Once a basis function is chosen, the images can be rated using the VDE metric, and the parameters of the basis function can be adjusted according to the search algorithm. The search algorithms for VDE transfer function generation are discussed in the next section.

3.4 Search

In order to obtain the transfer functions with the highest ratings, a search in the parametric space of the basis functions is performed. A particular set of parameters is rated by generating the transfer function with the parameters, rendering a set of images from different viewpoints, and computing the VDE metric (see Figure 3.3). Compared to evaluating a function in terms of an arithmetic expression, this is expensive and limited by the performance of the graphics hardware. Hence, the goal of the search is to test as few parameter sets as possible to keep the search time within acceptable boundaries. A brute force search is already too expensive for two parameters resulting in 256^2 combinations and is out of question for four and more parameters with at least 256^4 combinations.

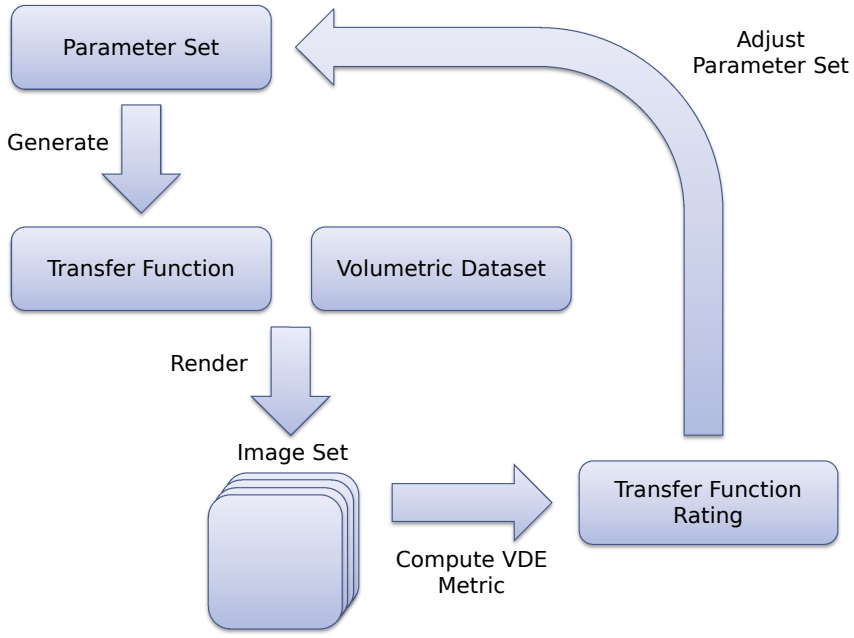
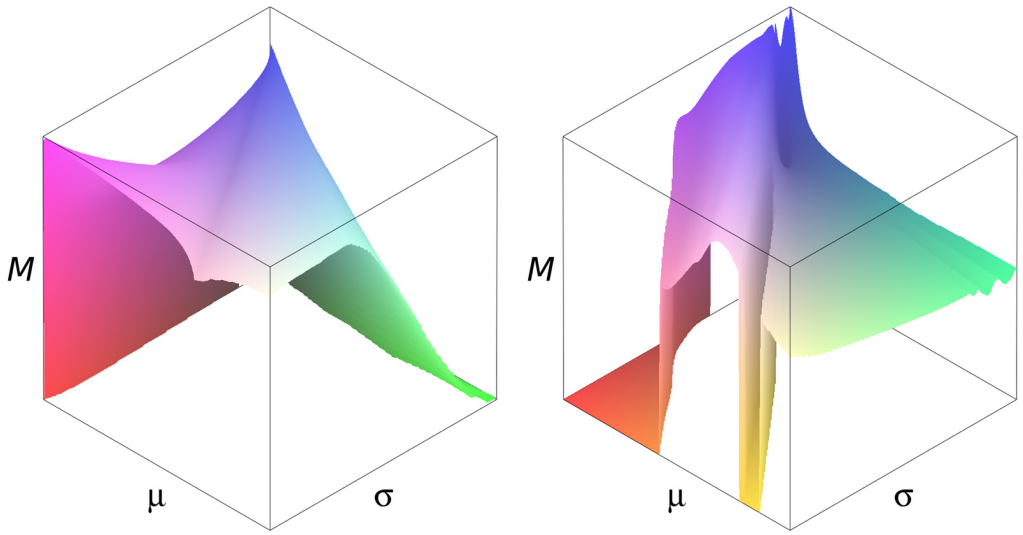


Figure 3.3: *The feedback loop for the transfer function search. The parameter set is used to generate the transfer function from a basis function. With the transfer function and the volumetric dataset a set of images from different viewpoints is rendered. Subsequently the VDE metric is computed using the image set and the parameter set is adjusted according to the search algorithm.*

When choosing a search algorithm various considerations have to be made. First and most important is the characteristics of the function to be searched. For VDE transfer function generation this is the VDE metric taking a set of images as input where this set itself is the result of a function defined by the renderer. The renderer takes the volumetric dataset and the transfer function generated using the

basis function and the set of parameters as input (see Figure 3.3). The resulting images heavily depend on the volumetric dataset and therefore the VDE metric heavily depends on the volumetric dataset. However, experiments indicate that the VDE metric shows a clear structure and is not random when plotted for a particular dataset and different parameter sets, which is important for a structured search. Further, the goal is not to find the single, global maximum but rather a set of local maxima or near maxima located as far away from each other as possible. Intuitively, this results in transfer functions generating images with high ratings that are distinct in what they reveal from the dataset. Exactly this is the goal of VDE transfer function generation. An example of function plots for different input parameters is shown in Figure 3.4.



(a) (b)

Figure 3.4: *Function plots of the VDE metric for a Gaussian basis function and different parameter sets (μ, σ) for the aneurysm dataset (a) and the skull dataset (b). Both plots show a clear structure and a limited number of local maxima.*

Considering the characteristics of the VDE metric as a function, gradient ascent is chosen for quickly finding a local maximum and simulated annealing as a randomized method for finding maxima in regions with high frequencies. We also suggest a combination of simulated annealing and gradient ascent as described subsequently. However, neither gradient ascent nor simulated annealing are suitable for searching the entire parametric space of the chosen basis function since it would require too many iterations taking too much time. Instead a number of seeds are used (see Section 3.5) and local searches are performed in parallel for the seeds.

3.4.1 Gradient Ascent

Gradient ascent or its counterpart gradient descent for finding the minimum are simple, first order optimization algorithms [Snyman, 2005] based on the gradient. Given a differentiable function $f : \mathbb{R}^3 \longrightarrow \mathbb{R}$ and starting from a position $\mathbf{x}$, steps proportional to the positive gradient are taken to approach the local maximum:

$$\mathbf{x}_{n+1} = \mathbf{x} + \gamma_n \nabla f(\mathbf{x}) \text{ for } n \geq 0 \tag{3.24}$$

where γ_n is a small enough factor such that $f(x_{n+1}) > f(x_n)$. Typically, the iteration stops when either a maximum number of iterations is reached or when $|f(x_{n+1}) - f(x_n)| < \epsilon$ for a small ϵ. γ can be set to a constant, or it can be computed using a line search. Line search yields a better convergence of the search but unfortunately it involves evaluating f for a number of values. Due to the computational cost of evaluating f (the VDE metric) it is better to set γ to a constant and accept a somewhat slower convergence. As the VDE metric cannot be differentiated analytically, the gradient is approximated by central differences similar to Equation 2.30.

3.4.2 Simulated Annealing

Occasionally the function plot of the VDE metric is not as smooth as in Figure 3.4(a) but contains high frequencies producing many local maxima as demonstrated in the green part of Figure 3.4(b). In this case gradient ascent starting from a coarsely set seed in the area would head straight for the next local maximum not considering other local maxima. Simulated annealing [Kirkpatrick et al., 1983] as a randomized method solves the problem stochastically without an entire search of the area of interest or the necessity to distribute finely grained seeds. Originally, simulated annealing was proposed for finding the minimum but since VDE transfer function generation requires the maximum, the algorithm has been adapted accordingly.

Given a function $f : \mathbb{R}^3 \longrightarrow \mathbb{R}$ with a starting position $\mathbf{x}$ and neighborhood $\mathbf{N}(\mathbf{x})$, simulated annealing moves to a random neighbor $\mathbf{x}' \in \mathbf{N}(\mathbf{x})$ according to a probabilistic criterion $P(M = f(\mathbf{x}), M' = f(\mathbf{x}'), T)$ where T is a temperature. The original proposition for P is:

$$P(M, M', T) = \begin{cases} 1, & \text{if } M' > M \\ e^{\frac{M'-M}{T}}, & \text{otherwise} \end{cases}. \tag{3.25}$$

The larger the difference $M' - M$ and the lower the temperature T are, the lower is the probability that the neighbor gets accepted. As with gradient ascent, the search is performed iteratively repeating conditional moves to random neighbors

until a maximum number of iterations is reached or $f(\mathbf{x})$ for the current $\mathbf{x}$ exceeds a certain threshold. The temperature T is derived from the number of iterations:

$$T = 1 - \frac{n}{n_{\max}} \tag{3.26}$$

where n is the current iteration and $n_{\max}$ is the maximum number of iterations. Consequently the probability of a neighbor with a lower value getting accepted diminishes with the number of iterations. Intuitively this means that at the beginning of the search the probability of not sticking to a local maximum but advancing to a different, possibly better maximum is high where towards to the end of the search, the nearest maximum is targeted.

With simulated annealing it is possible to find the best local maximum in a relatively small area containing many local maxima with a certain probability. However, simulated annealing is not as efficient as gradient ascent in large smooth areas as shown in Figure 3.4(a) since it considers only direct neighbors.

3.4.3 Combined Simulated Annealing/Gradient Ascent

Combining simulated annealing and gradient ascent is a way to profit from the advantages of both methods. In areas with low frequencies only gradient ascent is the method of choice as it quickly approaches the local maximum. Contrariwise in areas with high frequencies and a possibly large number of local maxima, simulated annealing is better because it tends not toward the next local maximum but finds the best local maximum with a certain probability. The idea is to figure out if the local area surrounding a starting point contains high frequencies. If this is the case, simulated annealing is used, otherwise gradient ascent can be utilized.

Given a starting seed, first simulated annealing is executed for a number of iterations. Apart from regular processing of the results, all values obtained during these iterations are recorded. Since simulated annealing only considers direct neighbors and does not make large steps, the recorded values are a good representation for the immediate neighborhood of the starting seed. In a next step the standard deviation of the recorded values is computed serving as an indicator for high frequencies:

$$\sigma = \sqrt{\frac{1}{|\mathbf{X}|} \sum_{\mathbf{x} \in \mathbf{X}} (\text{ø} - f(\mathbf{x}))^2} \tag{3.27}$$

where $\mathbf{X}$ is the set of all positions recorded during simulated annealing and ø is the average of all values processed. If the standard deviation is above a certain threshold indicating high frequencies, the search is continued with simulated annealing. Otherwise the search is continued using gradient ascent.

Now that the search is defined, the selection of the seeds as starting points for parallel local searches will be discussed in the next section.

3.5 Seeds

Given that the VDE metric generates function plots with several local maxima and that only one seed is considered as starting point, gradient ascent will head straight for the next local maximum. On the other hand simulated annealing will be able to find the global maximum and possibly several local maxima if the number of iterations is just high enough. However, this is not practicable for VDE transfer function generation because the required number of iterations is way too high taking too much time. Additionally, the goal is not to find the single, global maximum as quickly as possible but to find different local maxima in different areas of the definition space. The solution consists of a number of seeds distributed over the parameter space and local searches with a limited number of iterations. This allows for finding interesting local maxima in different areas of the parameter space with limited time requirements and can also be executed in parallel.

The easiest way is to distribute a number of seeds uniformly over the parameter space and rigidly perform the search for each seed. In a more elaborate way, additional seeds can be added in areas showing high potential and seeds can be removed from areas with low potential.

3.5.1 Selection

In the beginning a number of seeds is selected where the time required for the entire search correlates to this number. Choosing too few seeds raises the risk of missing interesting areas whereas choosing too many seeds takes a lot of time. Assuming no previous knowledge about the function plot of the VDE metric, a uniform distribution of the seeds is a good choice to raise the probability of finding local maxima throughout the entire definition space. However, introducing a level of randomness opens the opportunity for a second chance if the results are not satisfying. For this, the parametric space is subdivided into a grid with as many cells as seeds. Each seed is placed in an empty cell at a random position. If the search does not lead to satisfying results, changing the initial condition by re-initializing the seeds may deliver better results.

3.5.2 Subdivision

To further enhance the search, the potential of all cells is rated after each n iterations where n is an appropriate number. If a cell shows high potential, it is prioritized by a further subdivision and additional seeds are added to the subdivided cell. In contrast the search in a cell not showing potential is canceled prematurely. A way to rate the potential of a cell is to look at the seed as a random variable of the cell and compute the expected value. Unfortunately this requires the probabil-

ity distribution, which is not known at this point. A possible solution is to compute the probability distribution from the histogram of the values obtained during the search. However, it is obvious that in this case the expected value is simply the average ø of the obtained values. Since initial seeds comprising a low level are eliminated after the very first iteration as explained in the next section, the potential of a cell is judged rather by its variation than by its absolute value. For this the standard deviation σ from Equation 3.27 is used. If the standard deviation of a cell is high, it is likely that the cell contains high frequencies with many local maxima requiring more seeds to better search the cell. In contrast, if the standard deviation of the cell is very low, it is likely that the cell contains a uniform area with the same values everywhere. The transfer functions from this cell are likely to look the same and therefore the search can be canceled prematurely.

The decision whether a cell is further subdivided or canceled is made by comparing the standard deviation σ of a cell i to thresholds. If σ_i is higher than an upper threshold and a maximum number of subdivisions is not already reached, the cell is subdivided, whereas the cell is canceled if σ_i is lower than a lower threshold. If the dataset of such nature that each cell shows a high potential leading to a large number of subdivisions, it might be necessary to limit the number of subdivision by comparing the relative value κ of the standard deviation to thresholds:

$$\kappa_i = \frac{\sigma_i |C|}{\sum_{j \in C} \sigma_j} \tag{3.28}$$

where C is the set of all cells. In this case only the cells with the highest potential are subdivided and only the worst cells are canceled. The disadvantage is that fairly sophisticated cells might not be subdivided and therefore the search in this cells might be suboptimal.

By using subdivision it is possible to start with a limited number of seeds and quickly figure out the areas of interest and focus on them. The fact that in areas with high potential the number of seeds grows and the area to be searched for each individual seed becomes smaller allows to reduce the maximum number of iterations for the search making it faster.

3.6 Implementation

VDE transfer function search is implemented based on GPU volume ray-casting as described in Section 2.4. First, the volumetric dataset needs to be selected as well as the basis transfer function (i.e. a Gaussian) and the desired search algorithm (gradient ascent, simulated annealing or combined). Before the actual search can start, the seeds are initialized by putting them in a random position within their cell as explained in Section 3.5. The search itself can basically be

executed in parallel for all seeds. However, since the search involves computing the VDE metric requiring to render image sets, the bottleneck is the renderer in terms of the graphics hardware. As long as only a single graphics card is available, access to the graphics card is serialized and hence a large part of the entire search is serialized.

An important part of the search is the computation of the VDE metric. Since the focus lies on the transparency and the structure of the volumetric datasets and not so much on the colors, the VDE metric is computed on gray values only not considering harmonic colors. This heavily reduces the size of the alphabet for computing the entropy from 24 bit for a typical RGB pixel to 8 bit for a gray value. The transition from the color pixel to the gray value is achieved by the linear combination of $0.3 \cdot \text{red} + 0.59 \cdot \text{green} + 0.11 \cdot \text{blue}$.

After each iteration of each seed the result is evaluated and either stored in the list of the top results or dropped otherwise. The main criterion for a result to find entrance into the top list is the rating of the transfer function with respect to the VDE metric. Only if it is higher than the worst entry in the top list so far, the result is a candidate to be included. The second criterion is the similarity to existing entries. This is necessary due to the likelihood of many iterations around the global or a high local maximum producing equally looking images with high ratings. Without filtering these similar results, they may supersede interesting, different looking results from other areas of the search space. If the result is similar to an existing result in the top list, the result with the higher rating is kept and the other one is dropped.

The similarity of two results is computed by the Euclidean distance in the parametric space of the basis function. Given a basis function f, two results are similar if:

$$\|x - x'\| < \epsilon \text{ for } x, x' \in \text{dom}(f) \tag{3.29}$$

where x and x' are parameter sets and ϵ is a threshold.

An exception is the first iteration of all seeds. At the beginning a single iteration for all initial seeds is performed and the seeds are ranked as described above. Then the top 25 seeds are selected for further processing and all other seeds are dropped immediately. This is necessary because considering hundreds of seeds for the search takes far too much time.

The search is continued until all local searches started from the original or injected seeds are finished. Before presenting the results to the user a similarity index is computed between all results of the top list and the results are grouped according to this similarity index. In contrast to the similarity based on the Euclidean distance as a criterion to include or drop a result, the similarity index for grouping the results is computed in the image space from thumbnail images. The reason is that the results should be coherently grouped based on the thumbnail im-

ages presented to the user. First, the histograms of all thumbnails are computed:

$$G_i(x \in X) = \sum_{(m,n)\in t_i} \begin{cases} 1, & \text{if } t(m,n) = x \\ 0 & \text{otherwise} \end{cases} \tag{3.30}$$

where t_i is the i^{th} thumbnail of the set of thumbnails T and $t(m,n)$ corresponds to the pixel at position (m,n). X is the dynamic range of the computed gray value, which is typically 8 bit. Then the normalized similarity index is the average distance to all other thumbnails:

$$S_i = \frac{1}{|T||X|\max(X)} \sum_{j\in T\setminus\{i\}} \sum_{x\in X} |G_i(x) - G_j(x)| \tag{3.31}$$

Now the grouped results are presented to the user who can investigate them by clicking the thumbnails as depicted in Figure 3.5. The basic flow is shown in Algorithm 3.

3.7 Results and Discussion

To demonstrate how well VDE transfer function generation works, it was applied to a number of datasets. All experiments were performed on a Mac Pro 2.4GHz dual-Xeon with NVIDIA GeForce GTX 285 graphics. If not otherwise stated a viewport of 800^2 pixels was used.

VDE transfer function generation comprises a lot of parameters that can be adjusted to bias the VDE metric and to direct the search. However, a set of parameters constituting an optimal compromise between the quality of the generated transfer functions and the performance of the search was elaborated in countless experiments. Since this set of parameters proved to work very well, it was used for all experiments presented in this result section. This includes that for all results a combined simulated annealing/gradient ascent search was used. Experiments showed that gradient ascent alone is slightly faster than the combined search but the results are not as good. On the other hand, simulated annealing alone takes too much time to produce good results. This can be attributed to the fact that simulated annealing only considers neighbors and needs too many iterations in large, uniform areas to find a local maximum. An overview of the parameters is listed in Table 3.1.

The images presented in Figures 3.6, 3.7, 3.8, and 3.10 were composed as follows. First, VDE transfer function generation was applied to the respective dataset taking a couple of minutes to generate a set of 20 best transfer functions. Subsequently the transfer functions were presented to the researcher using the interface depicted in Figure 3.5. Finally, the images were synthesized by picking a transfer

Algorithm 3 Basic flow of the transfer function search

{——— Setup ———}
$V \leftarrow$ volumetric dataset
$f \leftarrow$ basis transfer function
$a \leftarrow$ search algorithm {returning adjusted parameters}
$R \leftarrow \{\}$ {empty result list}
$S \leftarrow$ initialize seeds {in terms of parameters}
while $|S| \neq 0$ **do**
  for all $s \in S$ **do**
    {——— Perform search iterations ———}
    $s \leftarrow$ perform search a with s and M {assume $M = 0$ if not yet initialized}
    $TF \leftarrow$ generate transfer function with f and s
    $I \leftarrow$ render image set with V and TF
    $M \leftarrow$ compute VDE metric with I
    {——— Evaluate search iteration ———}
    if $M > \min(R)$ **then**
      $d \leftarrow$ compute similarity criterion using R
      if $d <$ threshold **then**
        if $M >$ similar entry in R **then**
          Replace similar entry in R
        end if
      else
        $R = R \cup \{M\}$ {add current result to the result list}
      end if
    end if
    {——— Abort and subdivision criterion ———}
    $\kappa \leftarrow$ compute weighted average and standard deviation from last iterations
    if Abort criterion from a for s reached OR κ below lower threshold **then**
      $S = S \backslash \{s\}$
    else
      if $\kappa >$ upper threshold **then**
        $S = S \cup \{$seeds from subdivision$\}$
      end if
    end if
  end for
end while
{——— Prepare results ———}
for all $r \in R$ **do**
  Compute similarity index
end for
Group R according similarity indices
Present R to the user

function from the set of best transfer functions, choosing a single color, selecting a default set of lighting parameters, and render them with a GPU volume ray-caster (see Section 2.4). Choosing a color is necessary because VDE transfer function generation currently only considers the opacity channel. Though, choosing a single color in a color picker widget is a matter of a few seconds. If the dataset originates from a CT scan it is also possible to apply a default color map for the Hounsfield scale. The images in Figure 3.9 were composed in the same way except that the dataset was pre-segmented and a different color was chosen for each segment. Also the images in Figure 3.11 were composed in the same way except that two of the images were rendered using manually generated transfer functions from a participant of the user study.

Figure 3.6 demonstrates the effect of the different basis transfer functions. For this, a set of best transfer functions was generated for each type of basis transfer function and a good resulting transfer function was picked from each set. Remarkably, for the double Gaussian and the combined ramp Gaussian basis function, a steep Gaussian reflects the outer shell of the electron probability distribution of the neghip dataset. This is very well visible in the images as a semi-transparent surface.

Figures 3.7, 3.8, and 3.9 show that VDE transfer function generation is indeed able to produce distinct transfer functions that allow for insight into each facet of a dataset. Once the skin of the feet is rendered completely opaque hiding the bone structure, once only the bone structure is visible, and once the bone structure is visible with the skin rendered semi-transparent. The chest dataset is also a very good example. In the left image the skin is rendered semi-transparent revealing the trachea, the lungs, and the bronchi while completely omitting the skeleton. Contrariwise, in the right image, only the skeleton is visible and the lungs are entirely omitted. Examples for further datasets are shown in Figure 3.10. Basically, VDE transfer function generation delivered viable transfer functions for all tested datasets even though for some datasets they might not be as satisfying as desired (i.e. the knee dataset).

Table 3.2 presents an overview of the timings required for generating the set of best transfer functions for different datasets and basis transfer functions respectively. The time required for generating a set of best transfer functions is directly related to the total number of iterations executed and the performance of the GPU volume ray-caster. Computing the VDE metric on the CPU once the images are rendered is negligible. The total number of iterations is primarily dependent on the selected basis transfer function and the structure of the dataset. It is a priori much larger for the double Gaussian and ramp Gaussian basis functions due to the extended parameter space and the increased number of initial seeds. Additionally, if a dataset has a complex structure preventing a premature end of the search and triggering a lot of subdivisions, the number of iterations is further increased.

Parameter	Value	Explained in Section
Number of viewpoints	6	3.2
Noise term drop-off window: a_σ	14	3.2
Noise term drop-off window: b_σ	7	3.2
Noise term drop-off window: a_o	14	3.2
Noise term drop-off window: b_o	7	3.2
Noise term standard deviation patch size	20 x 20	3.2
Noise term pixel deviation patch size	10 x 10	3.2
Coverage term drop-off window: b_θ	30	3.2
Gaussian parameter range: a	$1/2$	3.3
Gaussian parameter range: b	$1/3$	3.3
Ramp parameter range: $b - a$	$1/2$	3.3
Ramp parameter range: c	$1/3$	3.3
Double Gaussian parameter range: a	$1/5$	3.3
Double Gaussian parameter range: b	0	3.3
Ramp Gaussian parameter range: $b - a$	$1/5$	3.3
Ramp Gaussian parameter range: c	0	3.3
Gradient ascent: γ	0.5	3.4
Gradient ascent: ϵ	0.01	3.4
Gradient ascent max iterations	50	3.4
Simulated annealing abort threshold	7	3.4
Simulated annealing max iterations	100	3.4
Initial number of seeds for Gaussian	216	3.5
Initial number of seeds for ramp	216	3.5
Initial number of seeds for double Gaussian	2500	3.5
Initial number of seeds for ramp Gaussian	2500	3.5
Seeds: σ lower threshold	0.6	3.5
Seeds: σ upper threshold	3.2	3.5
Iterations between subdivision/cancellation decisions	5	3.5
Number of injected seeds per subdivision	2	3.5
Similarity: ϵ	0.05	3.6

Table 3.1: *Parameter set for VDE transfer function generation. The parameter range for the basis transfer functions is the fraction in relation to the full range (see Section 3.3).*

Apart from the hardware, the performance of the GPU ray-caster is dependent on the viewport size and also on the dataset. A sparse dataset permitting a lot of empty space skipping or a dataset tending to opaque surfaces abetting early ray termination can greatly enhance the performance. Altogether these factors are responsible for the large differences of the timings between certain datasets. For generating the set of best transfer functions for the Gaussian and ramp basis function the average was 9:42 minutes and for the double Gaussian and ramp Gaussian basis functions 27:40 minutes. Even though this is not interactive or immediate at all, it is still fast compared to the time it takes to manually construct a similar set of transfer functions.

Dataset	Volume Size	Gaussian	Ramp	Double Gaussian	Ramp Gaussian	Figure
Neghip	64x64x64	13:29	9:55	23:19	46:49	3.6
Feet	512x512x250	22:13	12:23	35:45	58:00	3.7
Aneurysm	256x256x256	4:42	3:35	13:29	9:51	3.8
Chest	384x384x240	5:43	6:15	13:21	15:54	3.8
Engine	256x256x128	13:21	5:14	14:59	23:36	3.8
Pelvis	512x512x461	8:29	10:39	23:53	44:27	3.9
Head	128x256x256	19:45	12:21	48:58	45:13	3.10
Heart	512x512x75	8:28	7:19	26:32	30:28	3.10
Knee	379x229x305	4:33	5:58	18:24	22:50	3.10
Skull	256x256x256	19:58	10:37	37:34	38:42	3.10
Fuel	64x64x64	4:11	4:04	6:32	10:13	n/a

Table 3.2: *Timings for VDE transfer function generation for different datasets and basis transfer functions respectively. All times are in minutes.*

To underline the usefulness and effectiveness of VDE transfer function generation we conducted a small user study as well as a survey. The hypothesis of the study is that VDE transfer function generation delivers a set of transfer functions which make at least as many features of the respective dataset identifiable as a similar set created by a user in the same amount of time. Identifiable means that large parts of a particular feature are recognizable. It is not distinguished whether the feature is rendered opaque or semi-transparent. 11 undergraduate and Ph.D. computer science students with at least basic knowledge in computer graphics and visualization took part in the study. They were asked to manually construct transfer functions for the engine (Figure 3.8), chest (Figure 3.8), and knee (Figure 3.10) dataset using a Gaussian basis function with the goal of revealing as many features as possible. They had a time limit equal to the time required for the

automatic generation of the transfer function set for the respective dataset. Unfortunately the available machines for conducting the study were not state-of-the-art and therefore the interactivity was somewhat affected when designing the transfer functions for the chest and knee dataset.

Table 3.3 shows the results from the study. On average the automatically generated transfer functions make more features identifiable than the transfer functions created by the study participants and therefore the hypothesis of the study is true. We also investigated if the transfer functions from the study participants could detect features that were not revealed by the automatically generated transfer functions at all, but we were not able to find any. Contrariwise, the cartilage between the sternum and the ribs in the chest dataset becomes clearly visible when automatically generated transfer functions are applied but is not revealed using the transfer functions from many study participants (compare Figure 3.11). It was interesting to notice that the study participants often created transfer functions resulting in opaque surfaces when rendered, whereas the automatically generated transfer functions tend to transparent surfaces (also see Figure 3.11). Transfer functions resulting in images with opaque surfaces may be preferred by some users.

Number of identifiable features	Engine		Chest		Knee	
	ø	σ	ø	σ	ø	σ
VDE transfer function generation	15	0	10	0	7	0
TFs constructed by study participants (11)	13.0	2.7	7.0	1.6	6.4	0.9

Table 3.3: *User study for VDE transfer function generation. The table shows the number of identifiable features for the respective dataset. The number in brackets denotes the number of participants.*

In addition to the user study we also carried out a survey. Resulting images from automatically generated transfer functions for the chest (Figure 3.8), engine (Figure 3.8), fuel, and feet (Figure 3.7) dataset were presented to the participants. They were asked how useful they consider VDE transfer function generation for the respective dataset.

Table 3.4 shows the results of the survey. Members of the own laboratory are listed separately since their opinion might be biased. For the other participants the overall average is 4.2, 4.0, 2.3, and 3.6 for the chest, engine, fuel, and feet dataset respectively where 1 corresponds to ”not useful at all” and 5 corresponds to ”very useful”. The results indicate that the participants consider VDE transfer function generation useful for all datasets with the exception of the fuel dataset.

This may be attributed to the fact that the fuel dataset is very simple making it easy to manually construct a transfer function.

	Chest		Engine		Fuel		Feet	
	ø	σ	ø	σ	ø	σ	ø	σ
CS student (7)	4.1	0.38	4.0	0.82	2.1	3.00	3.6	0.79
Computer scientist (2)	4.5	0.70	4.0	0	2.5	0.70	3.5	0.70
Visualization expert (1)	4.0	0	4.0	0	3	0	4	0
Laboratory members:								
CS student (1)	5.0	0	5.0	0	4.0	0	5.0	0
Computer scientist (5)	4.2	0.84	4.4	0.89	3.2	1.30	4.4	0.89
Visualization expert (1)	5.0	0	5.0	0	4.0	0	5.0	0
Domain expert (1)	5.0	0	4.0	0	3.0	0	4.0	0

Table 3.4: *User survey of VDE transfer function generation. Users were asked how useful they consider VDE transfer function generation for the respective dataset where 1 corresponds to "not useful at all" and 5 corresponds to "very useful". The number in brackets denotes the number of participants. Members of the own laboratory are listed separately since their opinion might be biased.*

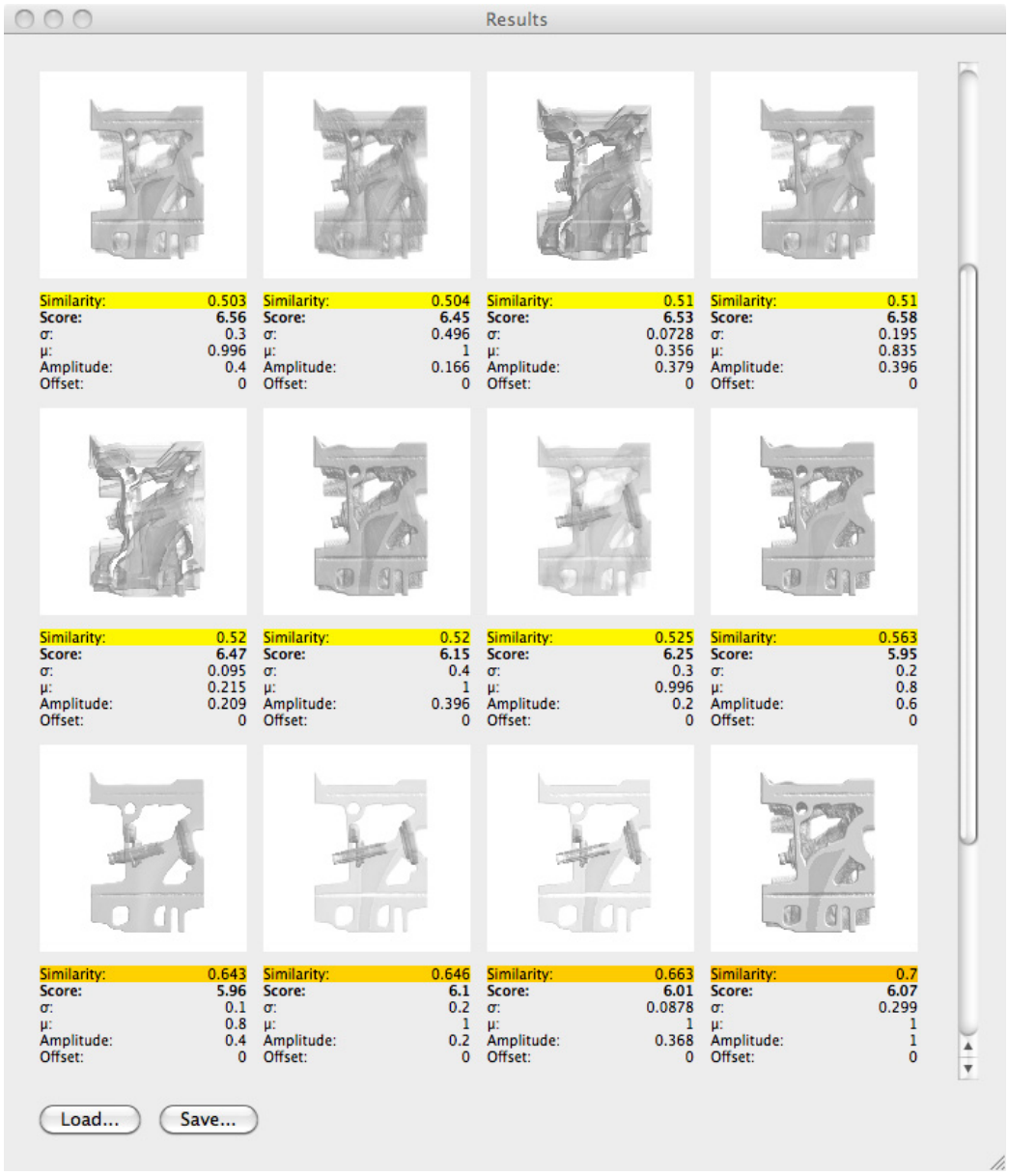


Figure 3.5: *The user interface for browsing the top results of the transfer function search. Each transfer function is presented to the user with a thumbnail image, the score, the similarity index, and the parameter set. The user can select a transfer function by clicking the respective thumbnail.*

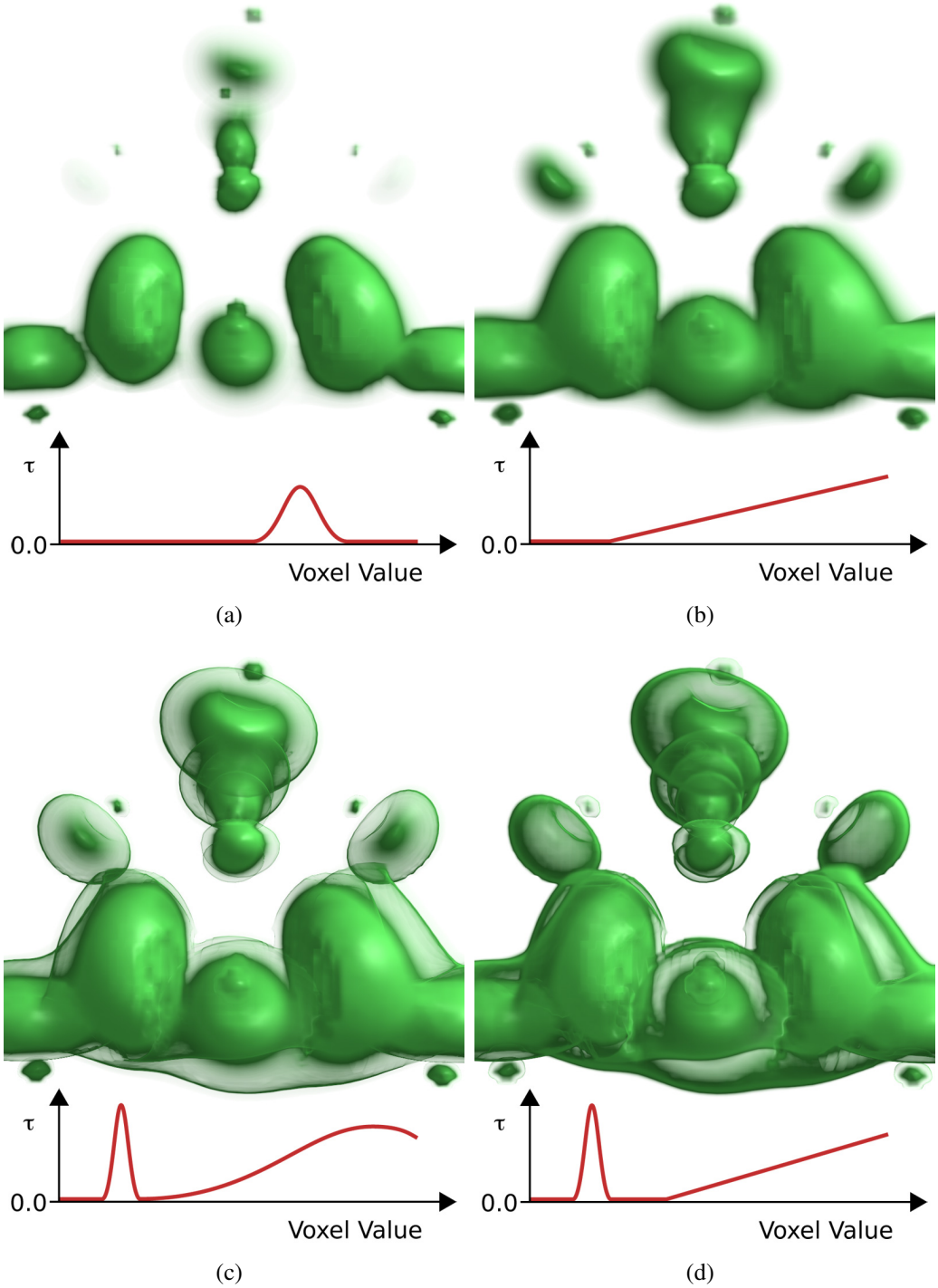


Figure 3.6: *Impact of the selected basis transfer function on the generated transfer functions. All images are rendered with a transfer function picked from the set of best transfer functions where (a) is based on a Gaussian, (b) on a ramp, (c) on a double Gaussian, and (d) on a combined ramp Gaussian. The semi-transparent surface from the steep Gaussian in (c) and (d) is visible very well. Dataset courtesy of SUNY, Stony Brook, NY, USA.*

Figure 3.7: *Different insights into the feet dataset. The transfer functions for rendering these images have been picked from the set of best transfer functions where (a) is based on a ramp, (b) is based on a Gaussian, (c) is based on a combined ramp Gaussian, and (d) is based on a double Gaussian. As expected, the generated transfer functions focus on different features and parts allowing for insight into each facet of the dataset. Dataset courtesy of OsiriX.*

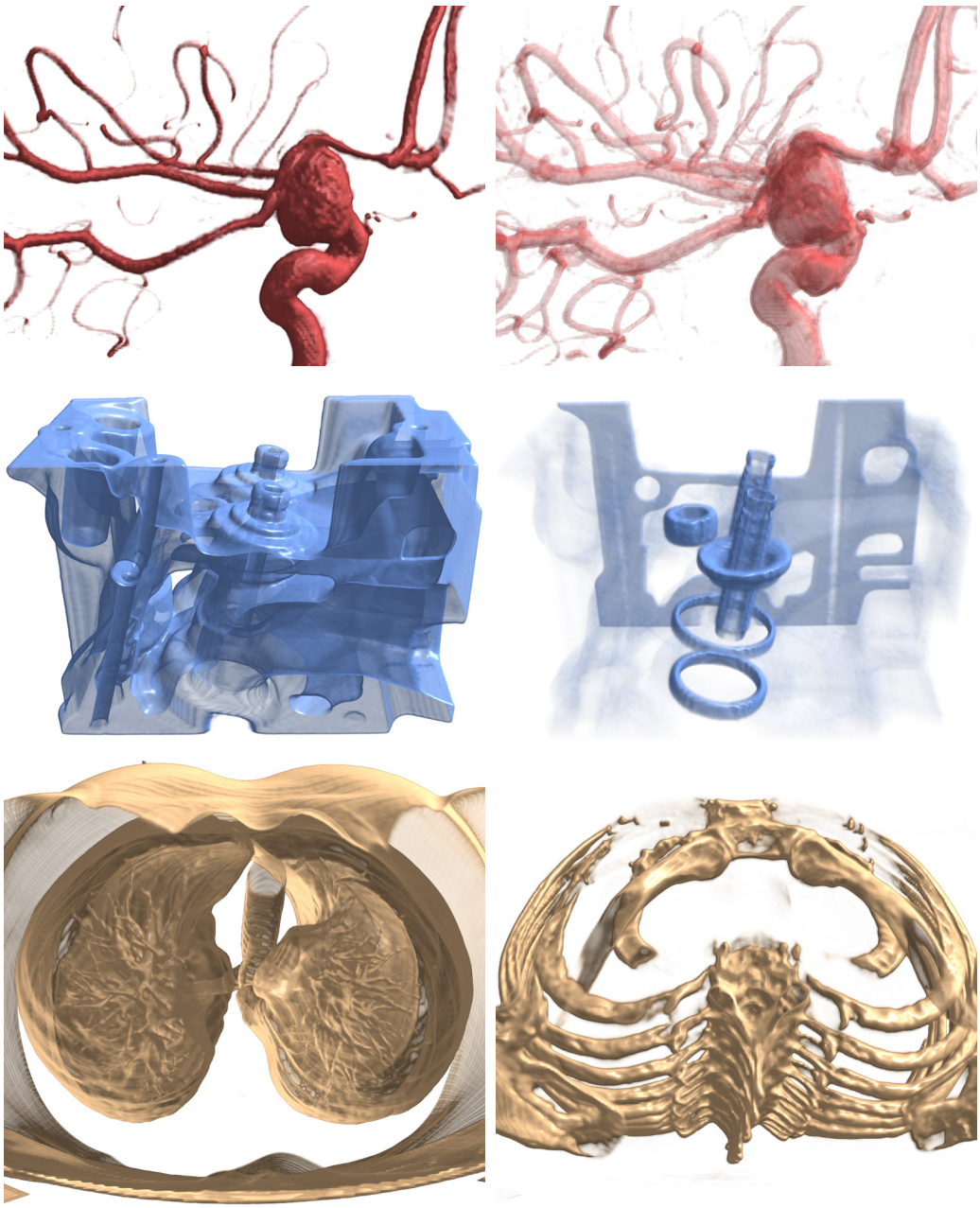


Figure 3.8: *Different insights into various datasets. The transfer functions for rendering these images have been picked from the set of best Gaussian transfer functions for the respective dataset. Datasets courtesy of Philips Research, Germany, General Electric, and the Department of Radiology, University of Iowa, USA respectively.*

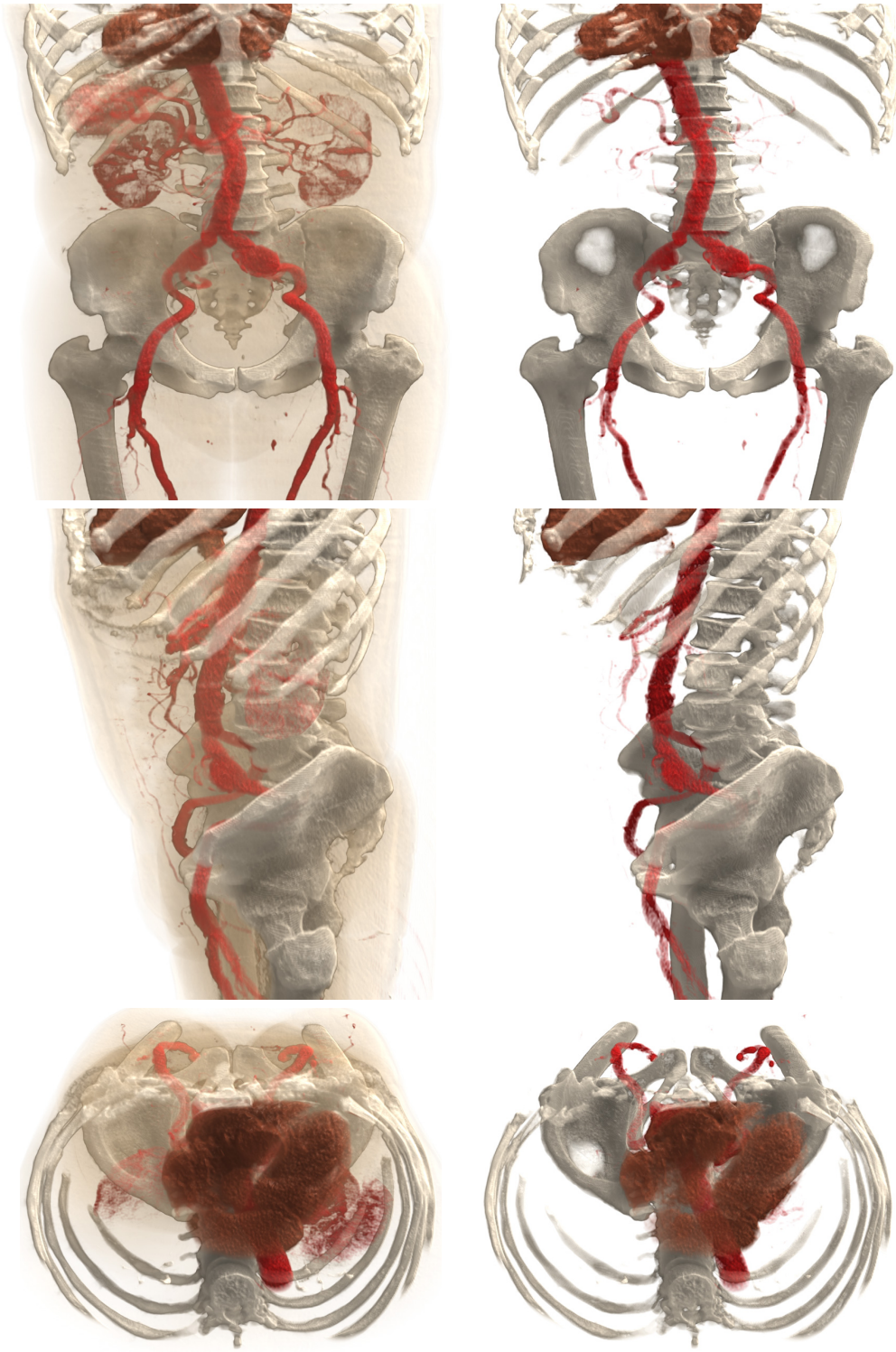


Figure 3.9: *Different insights into the segmented pelvis dataset. The transfer functions have been picked from the set of best ramp transfer functions. Additionally, a different color has been applied to each segment. Dataset courtesy of OsiriX.*

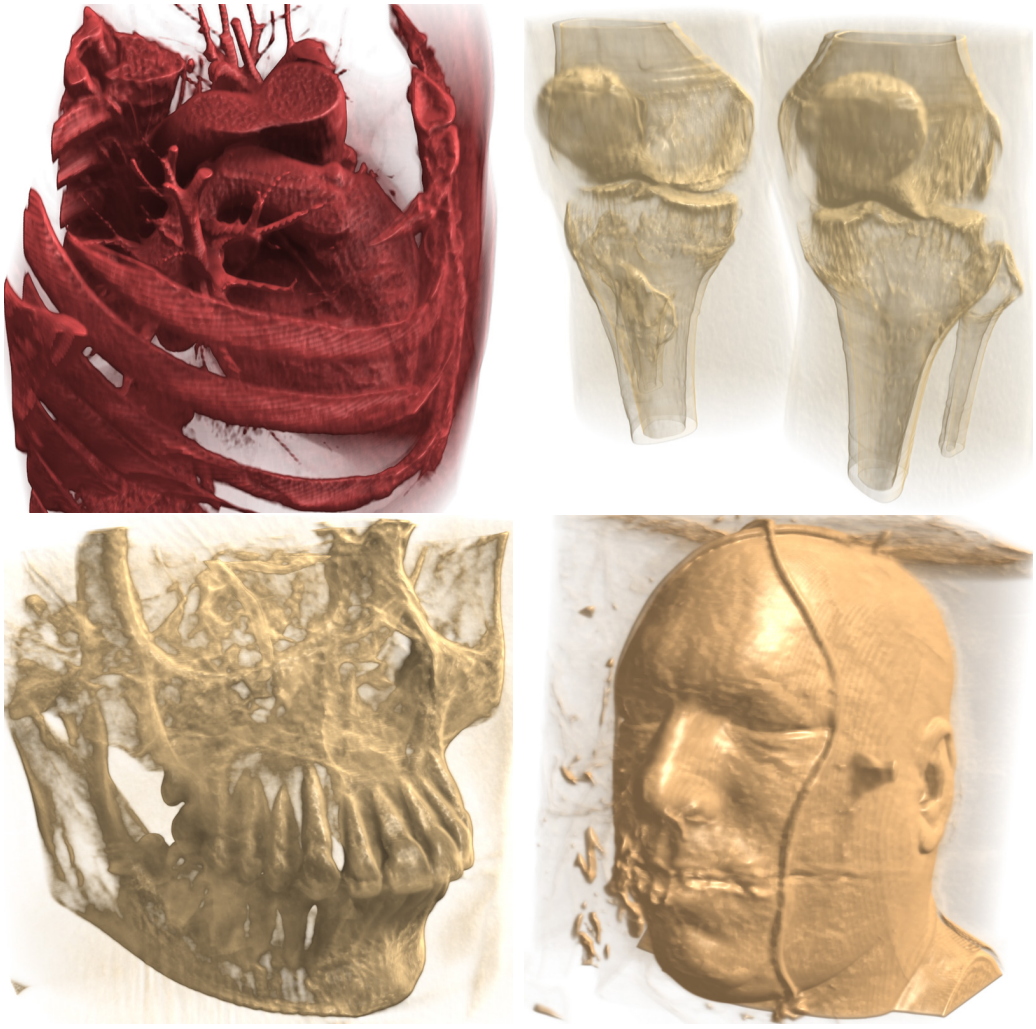


Figure 3.10: *Example images rendered with a transfer function from the set of best transfer functions for the respective dataset. Datasets courtesy of OsiriX, the Department of Radiology, University of Iowa, USA, Siemens Medical Solutions, Germany, and the National Library of Medicine, National Institutes of Health, USA respectively.*

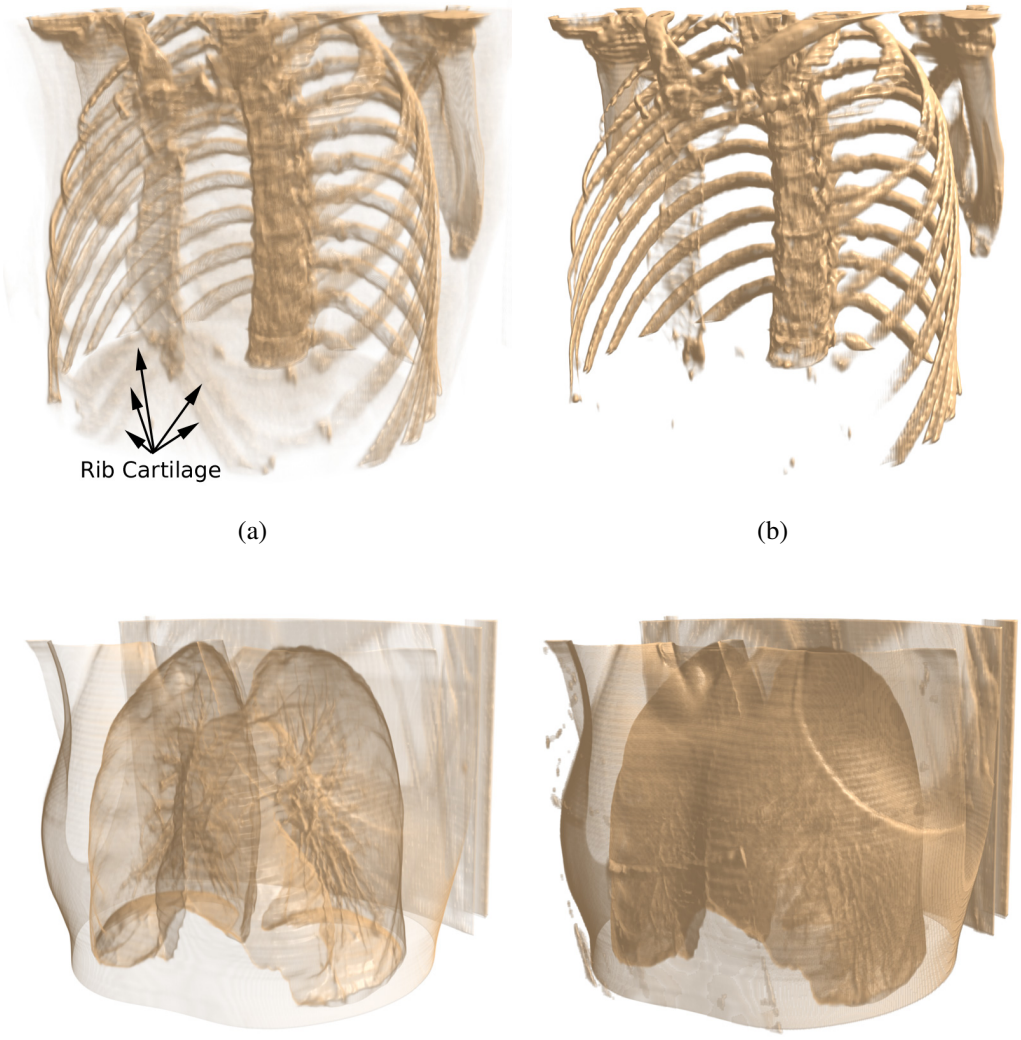


Figure 3.11: *Comparison of images rendered with transfer functions generated by VDE transfer function generation (a,c) and a user study participant (b,d). The transfer functions from the user study participant are similar to those from other study participants and show a tendency to more opaque structures. Hardly any study participant was able to generate a transfer function that uncovered the rib cartilage, which is clearly visible in the automatically generated transfer function (a). Dataset courtesy of the Department of Radiology, University of Iowa, USA.*

CHAPTER 4

EXTINCTION-BASED ILLUMINATION AND SHADING

4.1 Illumination and Shading Models

An important aspect of volume visualization methods is the way how illumination and shading are modeled and computed. Without an appropriate model the colors are displayed exactly the way they are fetched from the transfer function, leading to large areas featuring a uniform color. Within these areas it is not possible to recognize surface structures since reflections, highlights, and shadows are missing. The overall perception, and especially the depth perception are weak, not providing a satisfying impression of the scenario. It is therefore very important to apply an appropriate illumination and shading model during rendering. A sophisticated model can help to reveal details that otherwise would stay hidden. In this chapter we introduce a unified illumination and shading model featuring directional soft shadows, scattering, ambient occlusion, and color bleeding effects while achieving frame rates suitable for interactive applications.

4.1.1 Phong-Blinn

A common model for illumination and shading is the Blinn-Phong model [Phong, 1973; Blinn, 1977], which is also the standard model for the fixed function OpenGL pipeline [Shreiner et al., 2007]. Blinn-Phong is not limited to triangle rasterization but is also frequently used in volume visualization. Basically it consists of

an ambient term I_{ambient} representing diffuse, directionless ambient light, a diffuse reflection term I_{diffuse} representing the diffusely reflected light, and a specular term I_{specular} representing specular highlights. The overall light intensity I at a surface point is then:

$$\begin{aligned} I &= I_{\text{ambient}} + I_{\text{diffuse}} + I_{\text{specular}} \text{ where} \\ I_{\text{ambient}} &= k_A I_A, \\ I_{\text{diffuse}} &= k_D I_D \max(\mathbf{n} \cdot \mathbf{l}, 0), \text{ and} \\ I_{\text{specular}} &= k_S I_S \max((\mathbf{h} \cdot \mathbf{l})^s, 0). \end{aligned}$$

$k_{A,D,S}$ is the respective material property including color and the amount of reflected light, $I_{A,D,S}$ is the respective incoming light intensity, $\mathbf{n}$ is the normalized surface normal, $\mathbf{l}$ is the normalized light direction, $\mathbf{h}$ is the half vector between the view and light direction, and s is a shininess exponent. However, this standard model does not including advanced effects such as shadows, adjustment of the ambient light by occluders nearby (ambient occlusion) or color bleeding effects by light, reflected from colored objects to objects nearby.

4.1.2 Shadows and Advanced Effects

In the past many methods were proposed for modeling shadows and advanced illumination effects. A simple method for computing shadows is to cast so-called shadow rays from a sample point into the direction of the light source to determine if an occluder lies in between the sample point and the light source [Whitted, 1979]. But also very sophisticated illumination models such as radiosity [Goral et al., 1984; Sillion and Puech, 1994], global illumination [Dutre et al., 2002], and photon mapping [Jensen, 2001] were proposed taking many effects into account. They have in common their computational expensiveness and are hardly suitable for interactive applications. To incorporate shadows and advanced illumination effects into interactive applications, approximations are used that are not physically accurate but deliver visually plausible results. A good overview of advanced illumination models in the context of volume visualization, in particular GPU ray-casting, is provided in this work [Hadwiger et al., 2009].

Semi-transparent halos [Bruckner and Gröller, 2007] or directional occlusion [Schott et al., 2009] are computationally inexpensive and a good way to enhance the depth perception, but still do not provide shadows. Classic shadow maps [Williams, 1978] can be applied to volume visualization in limited scenarios to obtain hard shadows. To improve the performance for semi-transparent objects, deep-shadow maps [Ropinski et al., 2008a] use multiple opacity layers. Shadow maps are dependent on the transfer function and the light source and are typically evaluated for a single light source only. Half-angle slicing [Kniss et al., 2001;

Zhang and Crawfis, 2003] is able to produce more realistic soft shadows with small additional storage requirements, but also with the limitation to a single, directional light source. Using an additional shadow volume is another way to get soft shadows [Behrens and Ratering, 1998] but the particular method is limited to infinite light sources and requires recomputing the shadow volume upon each change of the light position.

Apart from these basic methods for directional shadows, advanced approaches have been suggested taking scattering (see Section 2.3) into account. For scattering it is required to integrate the incoming indirect light over all directions. This is prohibitively expensive for interactive applications and needs to be cleverly approximated. Kniss et al. [Kniss et al., 2002b] suggest cones from the sample points towards the light source based on half-angle slicing but with the limitation to a single light source. This method has been evolved [Ropinski et al., 2010] for GPU volume ray-casting at the price of an additional light volume requiring to be recomputed upon each change of the light position.

In addition to directional shadows, obscurance and ambient occlusion methods provide a simple way to approximate indirect global illumination by sampling a limited neighborhood [Méndez-Feliu and Sbert, 2008; Landis, 2002]. The basic idea is to determine how much a local neighborhood is occluded and adjust the amount of incoming ambient light accordingly. Screen-space methods [Shanmugam and Arikan, 2007; Díaz et al., 2010] are fast to compute and have a sufficient quality for opaque objects. Relying on the visible pixels' depth, they fail in complex scenarios and are inefficient for semi-transparent objects due to the limited depth information. This can be partly solved by enhancing screen-space ambient occlusion through depth-peeling [Everitt, 2001], adding additional depth information. Nevertheless, the computation of high-quality ambient occlusion requires object space methods at the cost of an additional volume, storing density or opacity values. This additional volume has to be updated upon each change of the transfer function. Updating the volume is slow in case the neighborhood of each voxel is individually sampled for achieving a good quality [Ruiz et al., 2008], or it is fast if aggregate values are considered [Díaz et al., 2010]. An approach based on per-voxel local histograms was introduced by Ropinski et al. [Ropinski et al., 2008b]. Transfer function and light source independent illumination is achieved by convolving local histograms with the current transfer function during rendering to obtain an environmental color of each voxel. Even though this is fast during rendering, the required preprocessing takes hours even for medium size models.

Color bleeding is often integrated in obscurance methods, as it requires only minor changes to ambient occlusion [Méndez et al., 2003]. Many ambient occlusion and translucency methods gather aggregated color information of the voxels' neighborhood together with the opacity, to use it for color bleeding effects [Ruiz et al., 2008; Ropinski et al., 2008b; Hernell et al., 2010]. Similarly, color bleed-

ing can be integrated in light scattering techniques as well [Kniss et al., 2002b; Ropinski et al., 2010].

4.2 Lighting with Additive, Exponential Extinction

In Section 2.3 the volume rendering integral is developed from the emission and absorption theorem into a discretized form featuring α-blending:

$$I(D) \approx I_0 \prod_{i=0}^{D/\Delta t} (1 - \alpha_i) + \sum_{i=0}^{D/\Delta t} C_i \alpha_i \prod_{j=i+1}^{D/\Delta t} (1 - \alpha_j). \tag{4.1}$$

This simplified version has been justified by the native support for α-blending in the hardware of the past. Nowadays, with the powerful, programmable GPUs, it is possible to implement any blending schema with hardly any performance impact at all. Hence it is feasible to use a discretized version of the original exponential extinction of the volume rendering integral as suggested in Section 2.3:

$$I(D) \approx I_0 e^{-\sum_{i=0}^{D/\Delta t} \tau(t_i)\Delta t} + \sum_{i=0}^{D/\Delta t} C_i \tau(t_i) \Delta t e^{-\sum_{j=i+1}^{D/\Delta t} \tau(t_j)\Delta t}. \tag{4.2}$$

Comparing Equation 4.2 to Equation 4.1 reveals that the entire extinction term is now the exponential of an order-independent sum. In this chapter we exploit the fact that it is an order-independent sum for defining our unified illumination and shading model. The basic premise is that any light occlusion and thus shadowing effects arise from the attenuation of light traveling or being scattered through the volume along a ray or within some specific region. Therefore, any light attenuation stems from some extinction factor $e^{-\sum_j \Delta t \tau_j}$ where the sum $\sum_j \Delta t \tau_j$ must be taken over a ray or region of the volume. The integration into ambient occlusion/color bleeding and directional shadows is discussed in the following.

4.2.1 Ambient Occlusion and Color Bleeding

Ambient occlusion is an approximated attenuation of diffusely reflected ambient light through occlusion. It is not physically accurate, but very useful and an effective approximation of the effect [Landis, 2002; Ropinski et al., 2008b; Hernell et al., 2010; Díaz et al., 2010].

The light term C in the volume rendering integral (Equation 4.2) includes an ambient term representing diffusely reflected ambient light. In the ambient occlusion lighting model, this term I_{AO} is not a constant but a function taking

the occlusion in the local neighborhood of a sample s into account. It represents the total amount of unoccluded incident light over a sphere Ω at s, where $I(s, v')$ denotes the incoming light at position s from direction v':

$$I_{AO}(s) = \int_{v' \in \Omega} I(s, v') dv'. \tag{4.3}$$

Instead of densely sampling the sphere Ω and tracing many shadow rays for $I(s, v')$, as shown by Ruiz [Ruiz et al., 2008] and Hernell [Hernell et al., 2010] respectively, we opt for a much faster approximation. Only a well-defined local neighborhood $\mathbf{N}(s)$ of s is considered for *local ambient occlusion*. We assume that for all samples s a constant ambient light intensity I_A is incident over the boundary $\partial\mathbf{N}(s)$. I_A is proportional to the sum of all lights, expressed by an ambient light term coefficient. Hence only the local light attenuation inside the neighborhood $\mathbf{N}(s)$ has to be considered. The local ambient occlusion is thus modeled by the distribution of the extinction τ in the neighborhood $\mathbf{N}(s)$ as

$$I_{AO}(s) \approx I_A \cdot e^{-\int_{t \in \mathbf{N}(s)} \frac{\tau(t)}{|s-t|^2} dt}, \tag{4.4}$$

where the inverse of the square distance accounts for the law of radial distance based light attenuation.

Color bleeding describes the phenomenon that the color appearance of a surface is locally affected by colored objects nearby [Méndez et al., 2003]. As this illumination effect is also primarily based on the local neighborhood of a sample point, it can be approximated in a similar way to ambient occlusion. For ambient occlusion only the extinction coefficient has been taken into account as an indicator for opacity at a particular position within the volume. For the estimation of color bleeding, the color also has to be taken into account as an additional parameter C_{RGB} depending on the transfer function. So Equation 4.4 can be reformulated to

$$I_{AO_{RGB}}(s) \approx I_A \cdot e^{-\int_{t \in \mathbf{N}(s)} \frac{\tau(t) C_{RGB}(t)}{|s-t|^2} dt}, \tag{4.5}$$

where $I_{AO_{RGB}}$ is a vector describing the intensity per color.

Since the summation of the exponential extinction coefficients in Equations 4.4 and 4.5 is order independent, it is possible to exploit highly efficient aggregate summation methods (aggregate query q), as described in Section 4.3. This yields results similar in quality at much higher speeds compared to individually sampling the neighborhood as demonstrated in Figure 4.1.

4.2.2 Directional Soft Shadows and Scattering

In the context of direct volume rendering, typically semitransparent surfaces and structures are displayed, partially obstructing other surfaces and structures. Thus

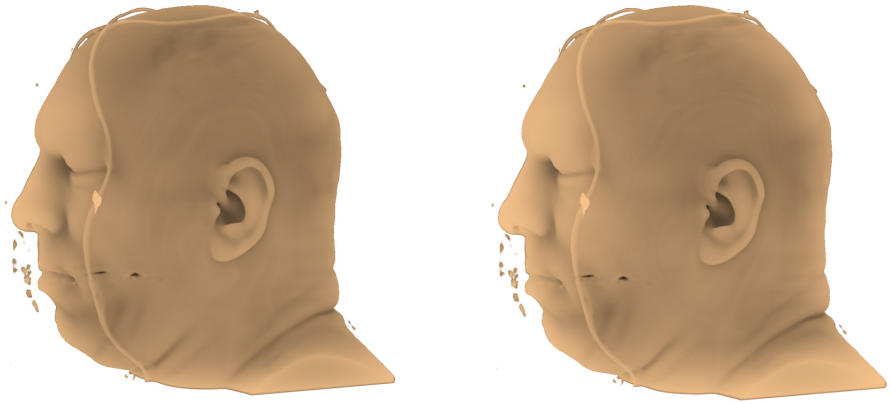


Figure 4.1: *Ambient occlusion with the neighborhood individually sampled (left) and aggregate extinction coefficients sampling (right). There is virtually no difference regarding image quality, but aggregate sampling is over 45 times faster (0.22s vs 10s). Dataset courtesy of the National Library of Medicine, National Institutes of Health, USA.*

for evaluating the volume illumination model the question is not whether a light source is obstructed or not but to what degree the light from this source is absorbed while traveling through the volume to the sample point in question. When using an extinction-based model as described above, this can be achieved by casting shadow rays to the light source, densely sampling along these shadow rays and summing up the weighted extinction coefficients. Applying the exponential function to this sum then results in a factor being a measure of how much light from this light source is attenuated before reaching the sample point.

One can imagine that densely sampling many shadow rays for each light source from each volume sample is already very expensive, multiplying the basic volume ray-casting costs by a large factor. Moreover, light scattering describes a process where non-uniformities in the (semi-transparent) medium force the traversal of light to deviate from the straight trajectory, caused by reflection of tiny particles (see Section 2.3). The volume rendering integral with scattering from Equation 2.22 makes:

$$I(D) = I_0 e^{-\int_0^D \tau(t)dt} + \int_0^D (E(s) + S(s,v)) e^{-\int_s^D \tau(t)dt} ds \tag{4.6}$$

where $S(s,v) = R(s,v,v')I(s,v')$, $R(s,v,v')$ is a bidirectional reflection distribution function (BRDF), and $I(s,v')$ is the incoming light reaching s from direction v'. When scattering occurs in multiple directions as it is the case in high albedo media, all directions v' have to be considered by integrating the scattering over the unit sphere.

According to Max [Max, 1995] it is an overkill to compute multiple scattering for most scientific visualizations, apart from being an elusive goal for interactive rendering. Instead of densely sampling shadow rays or computing multiple scattering, we suggest a very fast solution that produces similar effects with sufficient quality for most applications. The basic idea is not to cast a shadow ray from a sample to the light but a cone [Kniss et al., 2002b]. Sampling this cone not only yields the necessary extinction of the light on its way to the sample but also estimates the amount of light scattered towards the sample as a function of the distance and the neighborhood of the ray. A more sparsely occluded neighborhood is an indicator that there is more light scattered to the sample, thus supporting visually plausible soft shadow borders as presented in Figure 4.2.

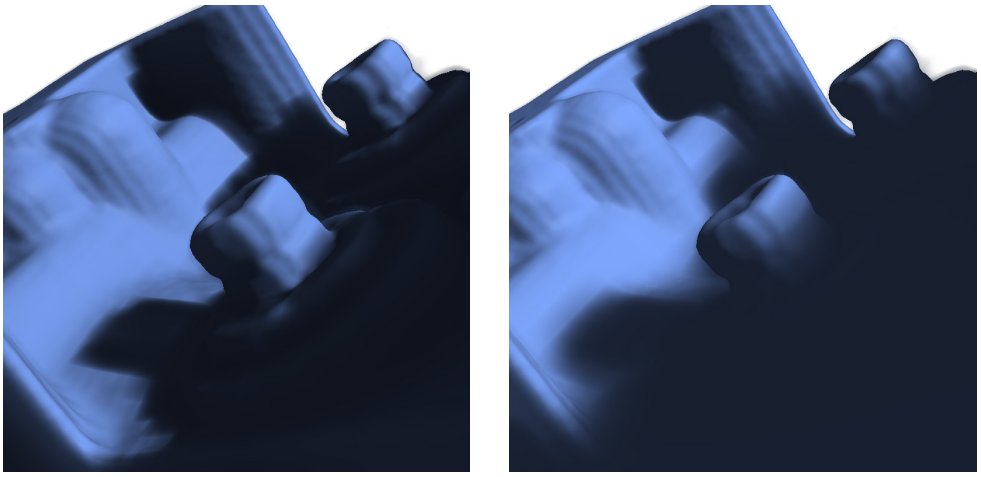


Figure 4.2: *The engine with classical shadow rays (left) and our soft shadows (right). Even though an entire cone is considered for soft shadows compared to a single shadow ray, the shadow computation is 40% faster (0.0079s vs. 0.0132s). Dataset courtesy of General Electric.*

The attractive feature of our method is that the shadow cone does not need to be sampled. To incorporate an estimate of the scattering term S into the discretized volume rendering integral of Equation 4.2, we evaluate shadow and scattering effects together by aggregating the extinction values within the shadow cone. Consequently the scattering term S and the emission term E are not treated separately but are combined into term C of the discrete volume rendering integral.

Commonly, the amount of light E_i reflected or emitted by a voxel is modeled as the sum of an ambient I_A, a diffuse I_D and a specular term I_S, where the ambient term is replaced by our ambient occlusion term I_{AO} introduced in the previous section. The diffuse and specular terms consist of the incoming light intensity I_L from all light sources, multiplied by material properties of the voxel

(i.e. color C_{RGB}) as well as angle-dependent diffuse and specular reflection factors respectively. To incorporate shadows, the light intensity I_L is further attenuated by a factor τ_L due to any occluders between the light source(s) L and voxel s: $\tau_L = e^{-\int_s^L \tau(t)dt}$. However, in our approach the integral is not only evaluated along a single shadow ray but over a cone Ψ towards the light source (see also Figure 4.3): $\tau'_L = e^{-\int_{t\in\Psi} h(t)\tau(t)dt}$ where h is a weighting function. Hence, the inclusion of scattering is the extension of τ_L to τ'_L as an attenuation factor of I_L, replacing the specific scattering term S, giving rise to a shadowed and scattered lighting term

$$I_{Sh}(s) \approx I_L \cdot e^{-\int_{t\in\Psi_s} h(t)\tau(t)dt}. \tag{4.7}$$

The key feature of our implementation is that the summation $\int_{t\in\Psi} h(t)\tau(t)dt$ is approximated by a series of aggregate extinction queries as described in Section 4.3. The weighting function $h(t)$ is the inverse of the aggregate query size W_q^{-1}. Due to the query size growing with the distance and the cone diameter, the influence of occlusion and scattering in these areas decreases rapidly.

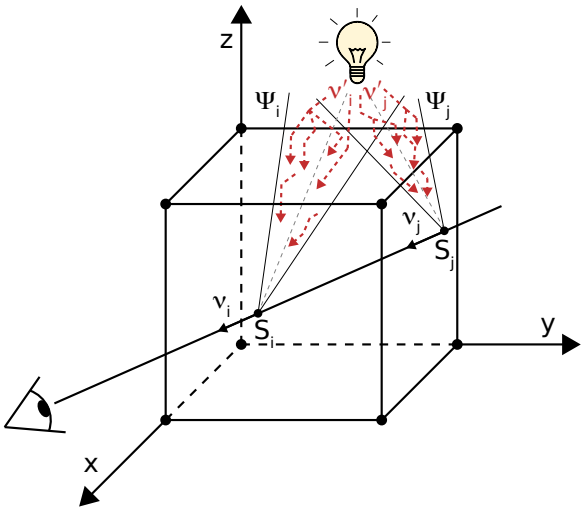


Figure 4.3: *Approximating scattering effects by considering cones Ψ instead of rays.*

The limitation of our approximation to scattering is the assumption that the scattering function R is constant for all directions and that the amount of scattering is basically proportional to the extinction coefficient of the medium. The rationale behind this is that the amount of light being absorbed or reflected directionally by the medium cannot be scattered isotropically or in a forward manner. Typically more light is absorbed or reflected directionally by denser media, especially when comparing gases to solids. In order to model medium specific scattering properties a separate transfer function would have to be applied. However,

adjusting the cone angle allows for a certain flexibility. A narrow cone approximates forward scattering, taking a limited range strongly into account whereas a wider cone approximates more isotropic scattering, taking a broad range into account but with far less influence of individual voxels. Scattering in relation with cones is also discussed extensively by Kniss et al. [Kniss et al., 2002b]. Generally, our approach works very well for typical applications of scattering in highly homogeneous media such as smoke (Figure 4.8) or a block of (wax-like) translucent material (Figure 4.9(d)).

4.3 Implementation

The basis for our implementation is a GPU volume ray-caster as described in Section 2.4. Algorithm 4 shows an overview of the rendering pipeline. Basically a so-called light cache (texture T) is computed containing the summation terms for ambient occlusion/color bleeding and the directional shadows in the different channels of the texture. During the ray-casting pass, these terms are fetched from the texture with a single lookup and used in the adapted illumination computation. In the beginning it has to be determined if either the transfer function or the light position relative to the dataset has changed. If this is not the case, the image can be rendered immediately, fetching the information for ambient occlusion/color bleeding and the directional shadows from the light cache texture. If the transfer function has changed, the light cache texture needs to be recomputed. For this, a 3D summed area table (SAT, texture S) [Crow, 1984] is constructed in a first step and then the light cache texture is computed in a second step. If only the light position has changed relative to the dataset, it is sufficient to recompute only the part of the light cache texture containing the values for the directional shadows.

4.3.1 3D Summed Area Table

For any illumination computation I_{AO} or I_{Sh}, as outlined in the previous section, we need to account for an attenuation factor $\tau_L = e^{-\int_\Omega \tau(t)dt}$ that will be multiplied with the light source intensity for ambient occlusion or directional soft shadows. In a discretized setup, this amounts to the computation of the sum of extinction coefficients $\sum_\Omega \Delta t \tau_j$, where the additive aggregation of extinction values τ_j is done over a voxel neighborhood $\Omega_A = \mathbf{N}(s)$ for ambient occlusion (and the ambient light I_A is modulated), along a ray $\Omega_L = \text{line}_{\text{voxel } s \text{ to light source}}$ for hard shadows and within a cone $\Omega_L = \Psi$ for soft shadows (and the light source intensity I_L is modulated). Taking ambient occlusion or shadowing into account, the reflected light C_i in Equation 4.2 of a voxel due to a light source is basically

$$C_i = I_{A,L} \cdot e^{-w_{A,L} \sum_{\Omega_{A,L}} \Delta t \tau_j} \cdot k \cdot C_{RGB}, \tag{4.8}$$

where the weighting function $w_A = r_q^{-2}$ is the inverse square of the radius of the aggregate query, and $w_L = W_q^{-1}$ is the inverse of the aggregate query size, and k simply represents a normal, gradient dependent local illumination model factor.

Algorithm 4 Overview of the rendering pipeline

```
1:  for each frame do
2:    if transfer function changed then
3:      Apply transfer function to volume V and store result to texture T
4:      /* ———————————————— SAT ———————————————— */
5:      Compute SAT from texture T by
6:      - Recursive doubling
7:      - Using ping-pong textures S, T
8:      Store SAT to S
9:      /* ———————————— Ambient/color bleeding factors ———————————— */
10:     for each voxel of S do
11:       Sample shells from S
12:       Sum up weighted shells
13:       Store sum to texture T
14:     end for
15:   end if
16:   if light source or transfer function changed then
17:     /* ———————————— Directional shadow factors ———————————— */
18:     for each light source L do
19:       for each voxel of S do
20:         Query cuboids towards light source L
21:         Sum up weighted cuboids
22:         Store sum to texture T
23:       end for
24:     end for
25:   end if
26:   /* ———————————————— Ray casting ———————————————— */
27:   for each pixel on screen do
28:     Compute entry and exit point for volume V
29:     Compute ray R from entry and exit point
30:     for each sample position P along ray R do
31:       Lookup volume V and apply transfer function
32:       Lookup texture T
33:       Evaluate illumination model
34:       Add contribution to pixel
35:     end for
36:   end for
37: end for
```

We implement the aggregation operation for the fast extinction summation over $\Omega_{A,L}$ using a summed area table (SAT) [Crow, 1984] of the extinction coefficients instead of using traditional expensive shadow ray generation and sampling. With a 3D SAT it is possible to derive the sum of all elements inside an arbitrary cuboid in constant time using at most eight table lookups. As shown below, we approximate the extinction summations over $\Omega_{A,L}$ by cuboid SAT queries.

Since the extinction coefficients are transfer function dependent, this SAT needs to be updated whenever the transfer function changes. However, fast SAT construction on the GPU [Hensley et al., 2005; Díaz et al., 2010] can be implemented based on the recursive doubling technique [Dubois and Rodrigue, 1977] using a logarithmic number of passes, allowing interactive transfer function changes (see Section 4.4). We use a render-to-3D texture approach which allows for a number of implementation synergies and avoids OpenGL-CUDA switches. Unlike Díaz et al. [Díaz et al., 2010] we are not using opacity values for the SAT but extinction coefficients.

In order to compute our illumination model two auxiliary 3D textures are used, one for the SAT, and another as a ping-pong texture during SAT generation becoming a light cache during rendering. These two textures can be of arbitrary size within the OpenGL limitations, depending on the desired quality/performance, and do not need to match the input volume resolution, see also Figure 4.11.

4.3.2 Ambient Occlusion and Color Bleeding

Remarkably, for approximating $I_{AO}(s)$ according to Equation 4.4, the extinction coefficient SAT can be effectively used. The discretized extinction coefficient summation $\sum_{\Omega_A} \Delta t \tau_j$ in Equation 4.8 is approximated by a series of cuboid shells as indicated in Figure 4.4, where the number and size of the shells can be varied. Hence, Ω_A is a set of cuboids Sh_i. For each shell, its aggregate sum of extinction coefficients can be obtained quickly by SAT lookups. A larger set of shells with varying diameters leads to a better image quality but requires more SAT lookups increasing the costs. According to our experiments as few as three shells are sufficient to reach an image quality hardly distinguishable from individually sampling a large neighborhood, as demonstrated in Figure 4.1. Only if the radius of Ω_A exceeds 10% of the radius of the entire dataset, more shells may become necessary. The use of cuboid shells is entirely different from Díaz' approach [Díaz et al., 2010], where the neighborhood is subdivided into eight adjacent octants preventing a distance based weighting.

For ambient occlusion/color bleeding, multiple shells are queried and accumulated as indicated in Figure 4.4. First, the innermost shell Sh_0 is queried from the SAT and weighted by the inverse square of its radius, $\tau_{Sh_0} = SAT(Sh_0) \cdot |r_{Sh_0}|^{-2}$. Iteratively all shells are accumulated by $\tau_{Sh_{i+1}} = \tau_{Sh_i} + (SAT(Sh_{i+1}) -$

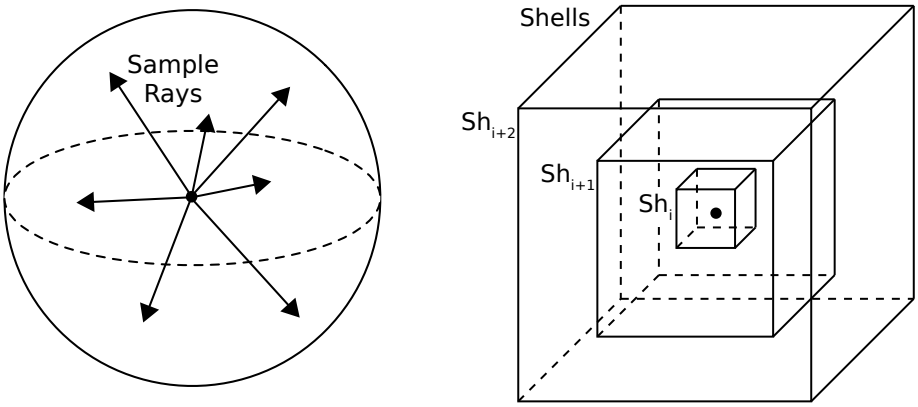


Figure 4.4: *Ambient occlusion computation by way of sampling the spherical neighborhood (left) versus SAT-based lookups (right). Compared to per-voxel sampling, the number of 3D texture fetches is an order of magnitude smaller using the SAT method.*

$SAT(Sh_i)) \cdot \left|r_{Sh_{i+1}}\right|^{-2}$ until the last shell is processed. The result of this summation is stored in the auxiliary 3D light cache texture.

Ambient occlusion is independent of the light position, but needs to be recomputed if the transfer function changes. The actual values for ambient occlusion are cached together with the values from the directional shadows in the 3D light cache texture. Consequently ambient occlusion in our solution comes at zero cost during rendering.

Of course Equation 4.5 for color bleeding can be computed similar to Equation 4.4 using a SAT that stores vectors τC_{RGB}. Since four values can be processed per operation with OpenGL textures, the SAT for τC_{RGB} can be constructed at the same time with the SAT for τ, and stored in the same 3D texture at no additional computation costs. The only downsides are the additional memory and memory bandwidth requirements compared to a single channel texture that would be used when constructing the SAT for τ only. However, on our hardware the additional memory bandwidth requirements do not harm rendering performance. The typical number of shells required for color bleeding proved to be the same as for ambient occlusion. An example is shown in Figure 4.5.

4.3.3 Directional Soft Shadows and Scattering

For the directional soft shadow illumination $I_{Sh}(s)$ according to Equation 4.7, two cases have to be differentiated. If the light sources are at a fixed position with respect to the dataset, as it is the case with the Cornell box model, the attenuation factors for directional soft shadows only have to be computed once and

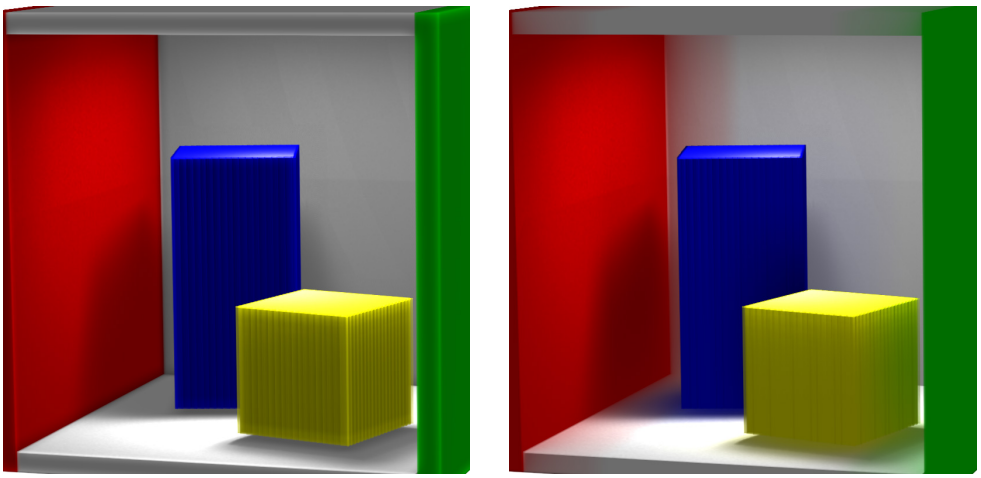


Figure 4.5: *The Cornell box with soft shadows and strong ambient occlusion (left), as well as color bleeding (right). Due to the fixed light source of the Cornell box, rendering with soft shadows and ambient occlusion/color bleeding enabled comes at near zero extra cost (one additional texture lookup). Dataset courtesy of the Cornell University, USA.*

are stored in the auxiliary light cache together with the terms for ambient occlusion/color bleeding. In this case, the total cost for evaluating our extinction-based illumination model during rendering consists of a single, additional texture lookup per sample having only a minor impact on the overall performance. If the light sources change their relative position with respect to the dataset when rotating, moving and zooming, then the occlusion factors have to be queried from the extinction SAT for every frame.

When needed, the attenuation factors for directional soft shadows, given by the discrete extinction coefficient summation $\sum_{\Omega_L} \Delta t \tau_j$ in Equation 4.8 over the sampling cone $\Omega_L = \Psi$, are computed by a render-to-3D-texture pass with the appropriate shader enabled. This shader approximates the attenuation cone Ψ for each voxel and light source by a series of cuboids. The primary cone axis is defined to be the coordinate axis with the smallest angle to the vector to the light source. The sampling points on the primary axis are given by a user defined sampling frequency and growth rate. The growth rate (growth of the cuboids) is the change of the frequency over the distance since further away a smaller sampling frequency may be sufficient. The cuboid queries are then derived from this primary axis sampling and from the projection of the query cone onto the primary-secondary axis planes as shown in Figure 4.6. Because the SAT inherently allows only axis-aligned lookups, deriving the primary and secondary axes is required. Choosing the axes in this way yields the best possible coverage of the cone with cuboids. With the cone covered by cuboids, the summation of the extinction co-

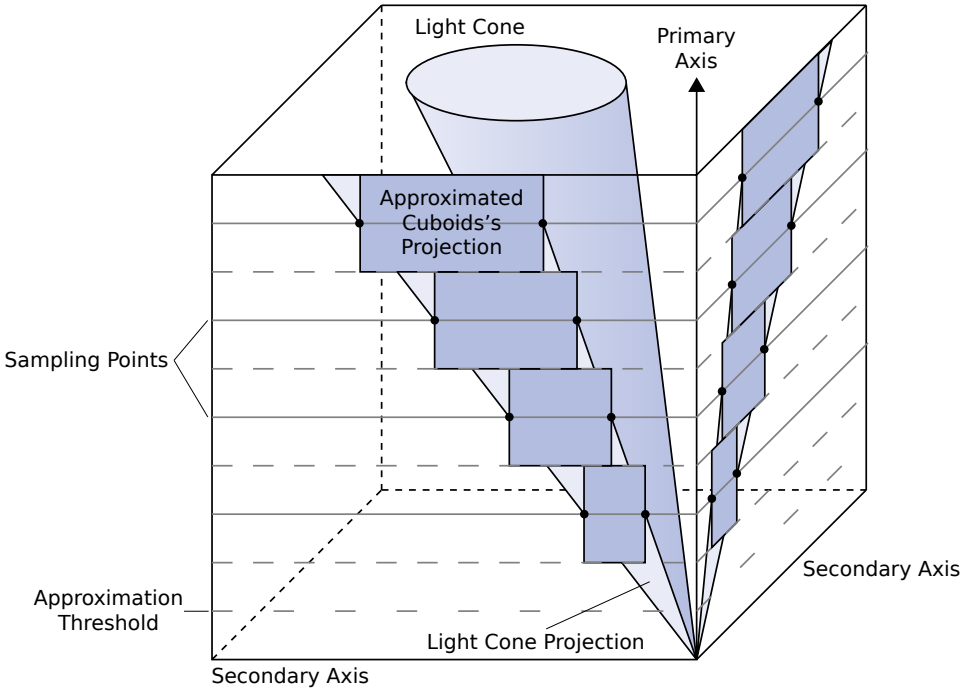


Figure 4.6: *The cone is approximated by a series of cuboids. The main axis is determined and the cone is projected onto the planes with the secondary axes. The intersections of the projections with lines parallel to the secondary axes through the sample points on the main axis define the cuboids.*

efficients can quickly be obtained by a few SAT lookups.

The shadow and scattering approximation by extinction SAT queries makes it very fast and flexible. The number of cuboids and the cone angle of Ψ can easily be varied, or the cuboids can be weighted differently using h in order to strengthen or weaken the effect. Approximating the cone by exploiting the SAT allows for soft, realistic looking, directional shadows at very low costs as shown in Figure 4.2. In contrast to the half angle slicing method by Kniss et al. [Kniss et al., 2002b], our solution can handle any type of light source as well as multiple light sources. It is also different from the method by Ropinski et al. [Ropinski et al., 2010] because we do not propagate illumination from the outside but compute the extinction of the light intensity for the voxels. We can therefore trivially handle light sources even within or on the border of the dataset without any additional effort. Multiple light sources can also be easily dealt with (see Figure 4.8 for multiple point and spot light sources inside the volume).

The angle of the cone Ψ, the number of cuboids for approximation (defined by a sampling frequency), the growth rate, and a weighting function are param-

eters that can be chosen freely according to the desired quality/performance and strength of the shadow effects (see Figure 4.10). Typically a few dozen lookups per cone and voxel are already sufficient to approximate the attenuation cone, compared to classical shadow rays where hundreds of samples are required to achieve a similar quality (see Section 4.4). Hence, even when computing these shadow terms for every frame, the performance impact is tolerable with respect to the achieved shading effects. To avoid duplicate shadow queries, the computed terms are stored in the light cache texture together with the terms for ambient occlusion/color bleeding.

4.4 Results and Discussion

All experiments have been performed on a Mac Pro with 2x Intel Xeon 2.4GHz processors, 8 GByte of memory, and a GeForce GTX 285 graphics card with 1 GByte of memory.

Compared to a Phong-Blinn-based GPU ray-caster, a ray-caster with our illumination model can produce realistic looking images with improved depth and occlusion effects (i.e. Figures 4.9, 4.10, 4.11). To ensure interactivity and responsivity, we use a 3D SAT enabling fast approximation of shadow cones with cuboids and ambient occlusion/color bleeding using cuboid shells. For each change of the transfer function, the extinction SAT and the ambient occlusion/color bleeding terms have to be recomputed. Every time the light source moves relative to the dataset or the transfer function changes, the terms for the directional shadows will be recomputed. During the actual ray-casting pass, one additional texture lookup per sample is sufficient to apply the illumination terms. Other approaches [Ropinski et al., 2008b] need two additional texture lookups for ambient occlusion, not considering directional shadows.

Table 4.1 demonstrates the interactive performance of our extinction-based illumination model. This includes computation of the SAT and the ambient factors once and the factors for directional shadows in every frame. The exceptions are the head, Cornell box, skull and bucky ball datasets where the factors for the directional shadows have to be computed only once due to the fixed light source(s).

The time required for constructing the 3D SAT for different sizes is 0.029, 0.067, 0.148 and 0.311s for 64^3, 128^3, 192^3 and 256^3 volumes respectively. Even though we do not use CUDA[1], the time is similar to the one reported by Díaz et al. [Díaz et al., 2010] for the 256^3 volume and is in fact much faster for smaller volume sizes. Moreover, our timings include the concurrent construction of the 3D SAT comprising the terms for color bleeding. Hence we can see that even for dynamic transfer function changes an interactive feedback can be achieved. The

[1] www.nvidia.com/cuda

Dataset	Volume Size	SAT Size	Shells	Cone Samples	Cone Angle	Lights [Dynamic/Static]	With Illumination	Without Illumination	Figure
Head	128 x 256 x 256	192^3	15	n/a	n/a	ambient only	111fps	143fps	4.1
Engine	256 x 256 x 128	64^3	n/a	50	$10°$	1/0	57fps	130fps	4.2
Cornell box	256 x 256 x 256	256^3	5	60	$16°$	0/1	133fps	161fps	4.5
Bucky ball	128 x 128 x 128	128^3	3	80	$2°$	0/1, 2, 3	55fps	62fps	4.8
Chest	512 x 512 x 75	64^3	3	50	$6°$	1/0	31fps	41fps	4.9(a)
Pelvis	512 x 512 x 461	64^3	5	50	$2°$	2/0	15fps	26fps	4.9(b)
Feet	512 x 512 x 250	128^3	3	50	$2°$	1/0	13fps	31fps	4.9(c)
Skull	128 x 256 x 256	128^3	5	50	$2°$	0/2	14fps	26fps	4.9(d)
Pelvis (comparison)	512 x 512 x 461	192^3	3	40	$1°, 3°, 5°$	1/0	3fps	30fps	4.10
Engine (comparison)	256 x 256 x 128	64^3	5	40	$2°$	1/0	26fps	49fps	4.11(a)
Engine (comparison)	256 x 256 x 128	128^3	5	40	$2°$	1/0	12fps	49fps	4.11(b)
Engine (comparison)	256 x 256 x 128	192^3	5	40	$2°$	1/0	5fps	49fps	4.11(c)
Engine (comparison)	256 x 256 x 128	256^3	5	40	$2°$	1/0	3fps	49fps	4.11(d)

Table 4.1: *Overall frame rates with and without extinction-based illumination for a 512^2 pixel viewport. Except for the head, Cornell box, skull and bucky ball, the directional soft shadows are computed dynamically for every frame.*

influence of the SAT size on the rendering is shown in Figure 4.11, demonstrating that the SAT size can easily be set at a fraction of the size of the volume dataset itself.

Table 4.2 shows the time required for computing the actual terms for ambient occlusion/color bleeding by approximating the neighborhood of every voxel with cuboid shells. The time is only dependent on the volume size and the number of shells but not on the neighborhood radius in contrast to explicit neighborhood sampling. Nevertheless, the last column shows the time for explicitly sampling a neighborhood of radius 7 ($= \frac{4}{3}\pi 7^3$ samples) to demonstrate the large time cost difference. Thus, even if not cached, ambient occlusion/color bleeding effects can be computed in real time for these volume models. The head in Figure 4.1 was rendered using a 192^3 ambient occlusion texture with 15 shells.

AO Shadow Texture Size	11 Shells	7 Shells	3 Shells	Sampled Radius=7
64 x 64 x 64	0.0039s	0.0030s	0.0022s	0.2099s
128 x 128 x 128	0.0202s	0.0126s	0.0065s	0.4441s
192 x 192 x 192	0.0606s	0.0348s	0.0147s	1.4280s
256 x 256 x 256	0.1382s	0.0791s	0.0307s	3.0818s

Table 4.2: *Time needed for computing the ambient occlusion/color bleeding terms for different light cache texture sizes and numbers of shells. The last column shows the time for explicitly sampling the neighborhood of radius $r = 7$.*

Note that combining SAT construction and ambient occlusion computation times for the example in Figure 4.1 results in a fairly low cost of only 0.073s. This highly interactive SAT and ambient occlusion computation avoids costly preprocessing [Ropinski et al., 2008b] to achieve transfer function independence.

Table 4.3 shows the time required for computing directional soft shadows for two different cuboidal cone approximation resolutions. The time required and the quality depend on the sampling parameters. The last column shows the time for a single, classical shadow ray with a sampling rate of 250 samples per unit (the side length of volume dataset). In fact, to get equally soft shadows, the cost of a single shadow ray would have to be multiplied by a factor ($\gg 1$) because sampling would have to be performed within an entire cone and not only on the ray.

For the engine in Figure 4.2 and the medical datasets in Figures 4.9(a) and 4.9(b) it is sufficient to use a 64^3 ambient occlusion/shadow texture with 50 samples per unit, consuming only 0.0079s/frame for shadow computations, to achieve a very good image quality. The medical dataset in Figure 4.9(c) was rendered using a 128^3 ambient occlusion/shadow texture with 50 samples, using 0.0385s/frame for

Soft Shadow Texture Size	50 Samples	20 Samples	1 Shadow Ray
64 x 64 x 64	0.0079s	0.0061s	0.0098s
128 x 128 x 128	0.0385s	0.0270s	0.0361s
192 x 129 x 129	0.1166s	0.0816s	0.0954s
256 x 256 x 256	0.2671s	0.1821s	0.2105s

Table 4.3: *Time required for computing directional soft shadows for different volume sizes and two different cone samplings. The last column shows the time for a single, classical shadow ray with a sampling rate of 250 samples per unit (the side length of the volume).*

shadow computations. The Cornell box in Figure 4.5 was rendered using a 256^3 ambient occlusion/shadow texture with 60 samples but updated only upon transfer function changes due to the fixed embedded light source. Figure 4.7 demonstrates the artifacts from the cuboids that become visible if only very few cone samples are used. Figure 4.10 shows the effects of different cone angles.

The scattering of light in smoke, thick fog or wax-like media with non-zero opacity is demonstrated in Figures 4.8 and 4.9(d), clearly showing the expected light shafts and diffusely scattered light propagation. Multiple different point and spot light sources inside the volume dataset and the corresponding illumination and shadow effects are demonstrated in Figure 4.8. Our solution can transparently and efficiently handle any such light sources (unlike e.g. half-angle slicing).

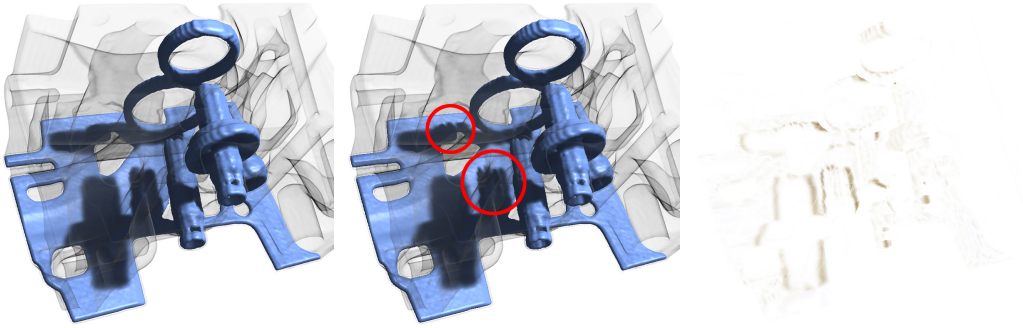


Figure 4.7: *As few as 12 cone samples are sufficient until cuboid artifacts become clearly visible for a SAT resolution of* 128^3*. The image on the right shows the difference between the two other images. Dataset courtesy of General Electric.*

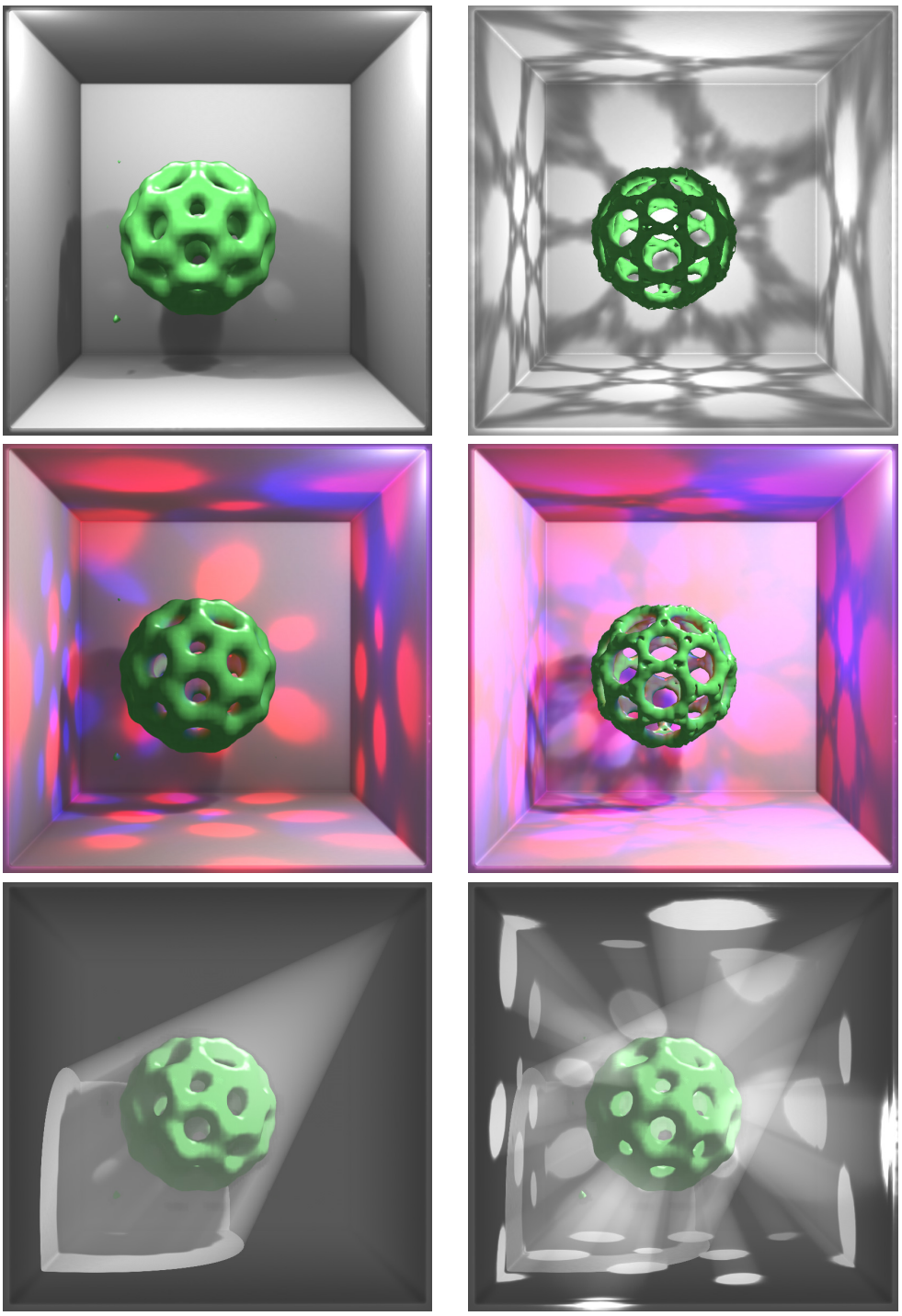
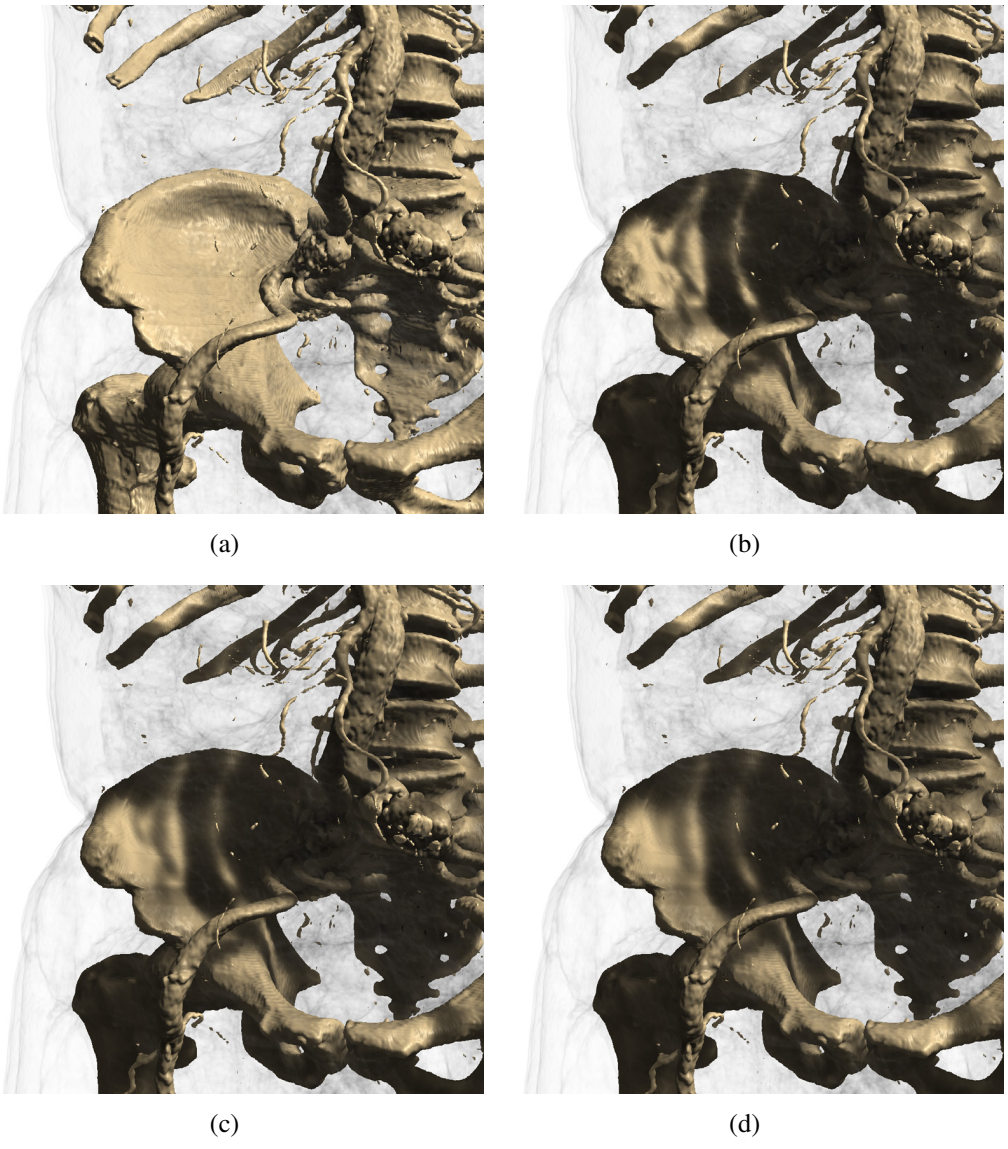


Figure 4.8: *Bucky ball in a box illuminated by up to three point lights inside the volume (two white point lights in top front box corners; one white light inside bucky ball; two examples of one white light in top-right box corner and two colored lights inside bucky ball; one white light in top-right box corner with smoke; one white light in top-right box corner and one light inside bucky ball with smoke). Dataset courtesy of AVS, USA.*

(a) (b)

(c) (d)

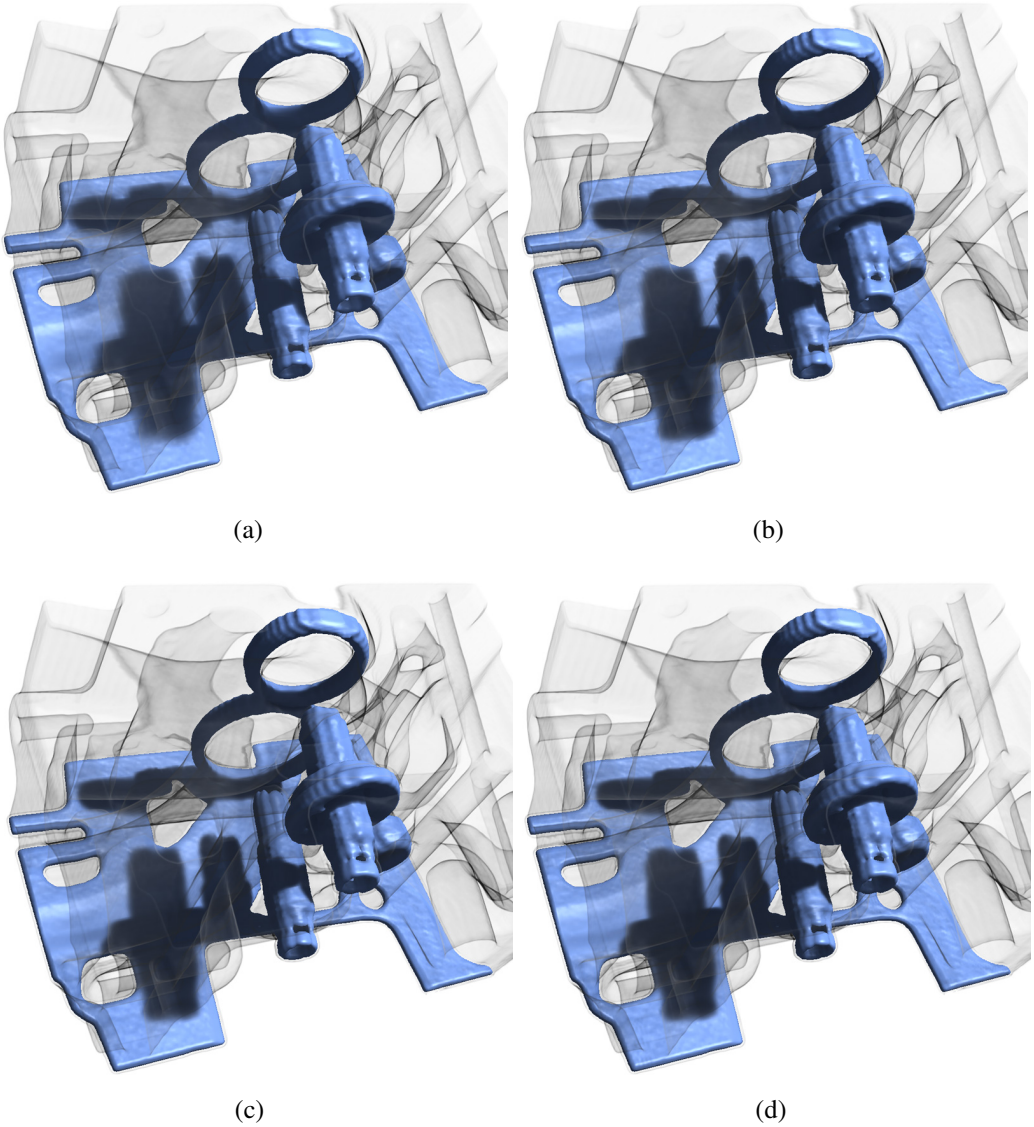
Figure 4.9: *Medical datasets rendered with extinction-based shading and illumination including directional soft shadows and ambient occlusion. The pelvis in image (b) is rendered using two lights and shows multiple shadows. The skull in image (d) is rendered in a thick fog or a block of translucent material with a point light source in the back scattering light through the medium and a spot light in the top left corner. Datasets courtesy of OsiriX and Siemens Medical Solutions, Germany, respectively.*



(a) (b)

(c) (d)

Figure 4.10: *Pelvis rendered with no directional shadows at all (a), and with different cone angles of* 1.0 *(b),* 3.0 *(c) and* 5.0 *(d) degrees, causing progressively smoother shadows. Dataset courtesy of OsiriX.*



(a) (b)

(c) (d)

Figure 4.11: *The engine dataset rendered with different SAT resolutions of* 64^3 *(a),* 128^3 *(b),* 192^3 *(c) and* 256^3 *(d) and 40 cone samples. Even lower SAT sizes will result in more blurred shadows but not expose conspicuous artifacts. Dataset courtesy of General Electric.*

CHAPTER 5

LAYERED SPLATTING

Splatting is an object space method for volume visualization introduced by Westover [Westover, 1989]. The basics of splatting are discussed in Section 2.5 including some of its limitations and the current state of art in splatting on the GPU. In particular it is shown that post-classified view-aligned sheet-buffer splatting delivers the best image quality but that exactly this variation of splatting is very expensive and consequently suffers from a poor performance. In this chapter we introduce layered splatting featuring the speed benefits of fast axis-aligned pre-classified sheet-buffer splatting while at the same time exhibiting display quality comparable to post-classified view-aligned sheet-buffer splatting. Sheet-buffer approaches typically split the interpolation kernel into several slabs to better approximate the volume rendering integral. Thus for every single voxel, multiple slabs have to be rasterized. In terms of performance the multiplied rasterization costs are a major bottleneck. Our new algorithm limits the number of required splatting operations to exactly one per voxel without losing the quality advantages of splatting multiple kernel slabs per voxel. We achieve this by applying a correction term based on the previous and consecutive sheet. Hence the sheets are not independent from each other anymore and that's why we call a sheet a *layer* and the method *layered splatting*.

Additionally, we enhance the quality by using a more accurate approximation of the volume rendering integral. Commonly, the extinction coefficient of the volume rendering integral is approximated by the first two elements of its Taylor series expansion to allow for simple α-blending as explained in Section 2.3. In our

approach we use the original, exponential extinction coefficient to achieve a better approximation. This is feasible by virtue of fast programmable GPUs facilitating any compositing schema at very low additional cost.

5.1 Performance Considerations

In order to develop layered splatting it is required to analyze the bottlenecks of the different variations of splatting, in particular sheet buffer splatting. Three of the major bottlenecks, sorting, rasterization and compositing are discussed as follows:

5.1.1 Sorting

Sorting is necessary to guarantee back-to-front or front-to-back traversal of the splats or sheets according to the simplified volume rendering integral featuring α-blending. Sheetless approaches such as the original splatting algorithm [Westover, 1989] have different traversal orders for every frame in which the view direction changes and thus need an expensive resort (for example heap sort [Heineman et al., 2008] in $O(n \log n)$) for every frame. Sheet splatting methods have the advantage that the individual voxels only need to be distributed to the different sheets whereas the order within a sheet is not important. The reason is that within a sheet the splats only interpolate the voxels independent of the order in terms of a convolution but are not blended together. Consequently it is sufficient to employ a cheap ($O(n)$) bucket sorting algorithm [Heineman et al., 2008]. When using axis-aligned sheet splatting, the distribution of the voxels to the sheets remains valid as long as a view direction change does not exceed a 45° angle. In this case the angle between another spatial axis and the view direction becomes smaller than the angle between the axis perpendicular to the sheets and the view direction and therefore the sheet orientation changes. The voxels have to be redistributed to the newly oriented sheets. For view-aligned sheet splatting, since the sheets are perpendicular to the view direction, the voxels have to be resorted for every change in view direction. This is a clear disadvantage over axis-aligned sheet splatting. Resorting on the CPU causes a lot of traffic on the bus because each time the whole geometry for the splats has to be transferred to the graphics card. For a medium sized dataset with dimensions 512^3 it means that without optimizations 537 million vertices have to be transferred to the GPU. Point sprites can diminish the amount of data sent to the graphics card since only one vertex is required per splat instead of several when using polygons. A recent method [Grau and Tost, 2007] does the resorting completely on the GPU eliminating the need to resend the geometry to the graphics card but limits the size of the dataset to the available GPU memory.

In our layered splatting approach we apply fast axis-aligned ordering such that voxel redistribution only has to be performed when crossing a $45°$ angle. Axis-aligned ordering is not only the way of layered splatting to relax the bottleneck of sorting but it is also prerequisite for the efficient computation and application of the correction terms as discussed in the next section. With normal axis-aligned sheet splatting, popping artifacts may occur when crossing a $45°$ angle and the sheet orientation changes. This is caused by the coarse approximation of the volume rendering integral being different for different sheet orientations. However, our layered volume splatting strongly abates the popping artifacts, which can be attributed to the use of more compact interpolation kernels and the interpolation correction term, see Section 5.2, as well as to the improved attenuation integration via the exponential extinction coefficient, see Section 5.3.

5.1.2 Rasterization

The splatting operation itself is a kind of 2D texture mapping including point sprites. Mostly it is the real bottleneck of volume splatting because texturing of millions of effective splats drives the current graphics cards to the rasterization limits. This applies especially when using sheet-buffered splatting in conjunction with an interpolation kernel that has a large support radius. The kernel then contributes to many sheets, and hence many footprints of kernel slabs have to be rasterized for a single voxel as shown in Figure 5.1(a). Neophytou et al. [Neophytou and Mueller, 2005] address this issue by using all color channels to splat four kernel slabs at a time. Although this solution is very fast, it has some weaknesses. For one, it only works since the normals are computed on-the-fly in a GPU program during compositing freeing the additional color channels commonly used for providing pre-computed normals. Obviously computing the normals on the fly comes at a premium. Furthermore, it is only well suitable using a Gaussian kernel because individual, pre-integrated kernel slabs can be conveyed from a base kernel by a single factor.

Our goal was to strictly have one single texturing operation per voxel including the possibility to provide a normal, either from the gradient volume or a gradient interpolation kernel. A gradient kernel is basically the partial derivate of the common interpolation kernel for the respective spatial axis. To achieve this, we switch from a splat centric view to a sheet centric view and, therefore, call a sheet a *layer*. A layer L_i contains the contributions of all interpolation kernels for which the corresponding voxel centers fall directly into L_i, plus correction terms from interpolation kernels from voxels in adjacent layers as illustrated in Figure 5.1(b). We define the invariant that only contributions from voxels centered in the current layer L_i are explicitly splatted. Contributions to L_i from voxels in adjacent layers are not explicitly splatted but approximated using a correction term.

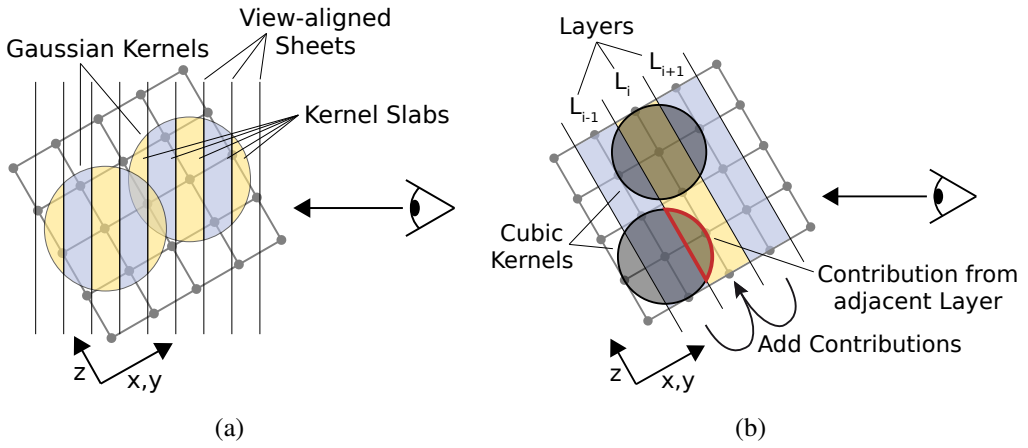


Figure 5.1: *(a) Sheet buffers are perpendicular to the view direction. Each Gaussian interpolation kernel with radius 2.0 spreads across five sheets resulting in five slabs per kernel at arbitrary position. All slabs are explicitly splatted. (b) The voxel grid with a layer overlay and two footprints of our cubic kernel. L_i is the currently processed layer. Contributions from footprints in the adjacent layers are approximated. They are not explicitly splatted as kernel slab footprints when using layered splatting.*

To minimize errors introduced by the correction terms, we no longer use a Gaussian interpolation kernel with radius 2.0 which may contribute to five layers. Instead we use a cubic interpolation filter with radius 1.0 that contributes to at most three layers. From a layer centric view this means that only voxels in the current layer plus voxels in the two adjacent layers must be considered. For a given layer L_i and its adjacent layers L_{i-1} and L_{i+1}, only the parts of voxels in L_i are rasterized as splats into the current layer's frame buffer. The missing contributions from adjacent layers L_{i-1} and L_{i+1} are accounted for on a per-pixel basis. This is achieved by accumulating contributions from L_{i-1} and L_{i+1} to the current layer L_i according to the ratio κ of the pre-integrated kernel intersecting L_i, see also Figure 5.1(b). Consequently the correction added consists of the contributions from the adjacent layers weighted by the correction factors κ.

The ratio κ may change for every voxel if they do not have the same positions perpendicular to the layers. This is typically the case with view-aligned layers, since in general the volume axes do not align with the view direction. To avoid this, we exploit axis-aligned layers to keep the relative positions of the voxels constant within a layer. Thus for a given blending kernel $s(x, y, z)$ we can pre-compute the correction factors $\kappa(x, y)$ once along the projection dimension since the kernels' intersections with adjacent layers L_{i-1} and L_{i+1} are constant for all voxels, as described in the following section.

5.1.3 Compositing

Using per-pixel post-classification and post-shading for high quality rendering, compositing and blending become crucial from a performance point of view, especially when the number of sheets or layers rises. It gets even worse if any kind of z-supersampling as in typical sheet based splatting is used to better approximate the volume rendering integral. Let us define the grid resolution of the volume being 1.0 and the distance between two sheets as 0.5. This effectively doubles the required amount of compositing operations but produces a higher quality image through a closer approximation of the volume rendering integral, particularly for low-resolution volumes. Huang et al. demonstrate this in their OpenSplat framework [Huang et al., 2000]. The compositing performance is basically independent from the effective number of voxels or splats as long as no special optimizations are made. Assuming classification and shading is done in a fragment shader, Neophytou et al. [Neophytou and Mueller, 2005] show how special OpenGL extensions can be used to optimize performance. Early z-culling and depth-bounds test extensions allow dropping of fragments that are not affected during splatting or which are already opaque in a front-to-back traversal. As we use a different extinction model, we cannot use the default OpenGL blending. Thus we compute blending within the fragment shader where classification and lighting takes place, and subsequently can take advantage from the same optimizations.

As z-supersampling is not required by our layered splatting approach, it benefits from a reduced number of compositing operations. This is feasible because of compact blending kernels, the interpolation correction terms accounting for adjacent layer contributions, and the improved attenuation factor from the exponential extinction coefficient. Excellent rendering quality is furthermore achieved due to high-resolution interpolation within layers and post-classification.

5.2 Cubic Reconstruction Kernel

Because of the discrete resolution of a sampled scalar (or vector) field, the gaps between the sample points must be interpolated for direct rendering and zooming as described in Section 2.1. In other words, a continuous 3D function has to be reconstructed from the available spatial samples. This reconstruction is not only crucial for quality but also for performance. The most common interpolation scheme is the (tri-)linear interpolation that is heavily used in ray casting based volume rendering. In the volume splatting context the Gaussian interpolation kernel is very popular. Apart from the superior quality of the Gaussian kernel over trilinear interpolation, there are some other properties that make it very attractive. The derivative of the Gaussian is essentially a Gaussian again. Further it can be considered spherically symmetric, making it independent from the view direction.

Frequently a Gaussian with a support radius of 2.0 is used:

$$s(r) := c \cdot e^{-2.0r^2} \text{ for } |r| < 2.0. \tag{5.1}$$

However, the Gaussian kernel does not satisfy very well the needs of layered splatting. As only the footprint in the layer where the voxel center lies is explicitly rendered, an error is introduced for every contribution of the kernel that lies outside of that central layer. A Gaussian with radius 2.0 contributes to four additional layers apart from the main layer where the voxel lies. Accordingly, it is better to use a kernel with a smaller support radius. In terms of performance this has an additional benefit. The individual footprint splats are smaller and thus fewer pixels have to be rasterized per footprint, further deferring the rasterization limit.

Interpolation kernel filters are discussed in Section 2.1. Instead of using one of the standard filters we have decided to introduce a special filter that optimally fits the needs of layered splatting. It is a separable filter defined by the one-dimensional function:

$$s_s := 1 - 3|x^2| + 2|x^3| \text{ for } |x| < 1.0. \tag{5.2}$$

A discussion with special focus on cubic interpolation filters can be found in the work by Mitchell [Mitchell and Netravali, 1988]. We chose this particular filter because it is zero outside a box with edge length 2.0 and subsequently spans exactly the three layers L_{i-1}, L_i and L_{i+1} in a regular voxel grid as indicated in Figure 5.1(b). Thus the error introduced by layered interpolation along the projection dimension z can significantly be reduced by a correction term. In fact, the integral of the 3D interpolation filter $s(x, y, z)$ of Equation 5.2 equals zero inside $[-1, 1]$. The correction factors $\kappa(x, y)$ for the correction term can be computed as follows:

$$\begin{aligned}
\kappa_{L_{i-1}}(x, y) &= \frac{\int_{-1}^{-0.5} s(x,y,z)dz}{\int_{-1}^{1} s(x,y,z)dz} = 0.09375 \\
\kappa_{L_i}(x, y) &= \frac{\int_{-0.5}^{0.5} s(x,y,z)dz}{\int_{-1}^{1} s(x,y,z)dz} = 0.8125 \\
\kappa_{L_{i+1}}(x, y) &= \frac{\int_{0.5}^{1} s(x,y,z)dz}{\int_{-1}^{1} s(x,y,z)dz} = 0.09375.
\end{aligned}$$

It shows that κ is independent from the position (x, y) due to the separable characteristics of the kernel. The final interpolated result for layer L_i can be obtained by first splatting the voxels centered in L_i, followed by κ-corrected accumulation of values from layers L_{i-1} and L_{i+1}. Figure 5.2 demonstrates the effect of the correction term. Note that in contrast to Gaussian interpolation the amount of pixels to be rasterized is reduced by a factor of 4.0 without loss of rendering quality (see also Section 5.5).

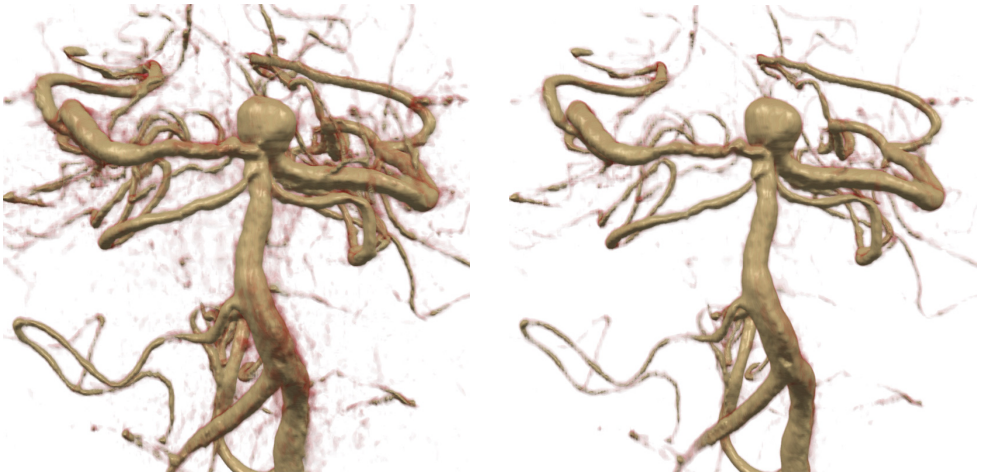


Figure 5.2: *Right image without using the interpolation correction term shows significant artifacts from missing contributions from parts of interpolation kernels overlapping adjacent layers. Significant amount of information is lost. Dataset courtesy of Viatronix Inc., USA.*

However, there is one disadvantage when using a filter that is not spherically symmetric as it is not independent from the view direction anymore. When splatting the footprints of such kernels, the view direction has to be taken into account and an appropriate footprint has to be selected. From a performance point of view it is not feasible to generate the footprints on the fly during splatting. However, pre-computing a set of oriented footprint images, storing them in some small texture cache, and choosing the most suitable in every situation solves the problem. Selecting the right footprint image can be done in a vertex shader program. Currently we use a set of 856 pre-computed footprint images. Each of these has a size of 64^2 pixels with 1 byte per pixel requiring a total of only 3.3 MByte texture memory. According to our experiments, the discretization of the view direction does not lead to visual artifacts.

5.3 Exponential Extinction

In Section 2.3 the volume rendering integral is developed from the emission and absorption theorem into a discretized form featuring α-blending:

$$I(D) \approx I_0 \prod_{i=0}^{D/\Delta t} (1-\alpha_i) + \sum_{i=0}^{D/\Delta t} C_i \alpha_i \prod_{j=i+1}^{D/\Delta t} (1-\alpha_j). \tag{5.3}$$

This simplified version forms the basis for the vast majority of volume splatting implementations since α-blending has been available natively in hardware for a very long time. It was the only chance to achieve interactive frame rates before programmable GPUs became available. Nowadays, with the powerful, programmable GPUs, it is possible to implement any blending schema, not only linear combinations of α-values, in a GPU program with hardly any performance impact at all. Hence it is feasible to use a discretized version of the original exponential extinction of the volume rendering integral as suggested in Section 2.3:

$$I(D) \approx I_0 e^{-\sum_{i=0}^{D/\Delta t} \tau(t_i)\Delta t} + \sum_{i=0}^{D/\Delta t} C_i \tau(t_i) \Delta t e^{-\sum_{j=i+1}^{D/\Delta t} \tau(t_j)\Delta t}. \tag{5.4}$$

Layered splatting incorporates exactly this form of the volume rendering integral replacing the fixed function α-blending of the graphics hardware by exponential extinction in the GPU program during compositing the layers. Figure 5.3 demonstrates the difference of layered splatting with α-blending and with exponential extinction. α-based extinction can handle complete opaqueness very well and may still be the choice if the focus is on iso-surfaces whereas exponential extinction shows the typical smooth transparency transitions.

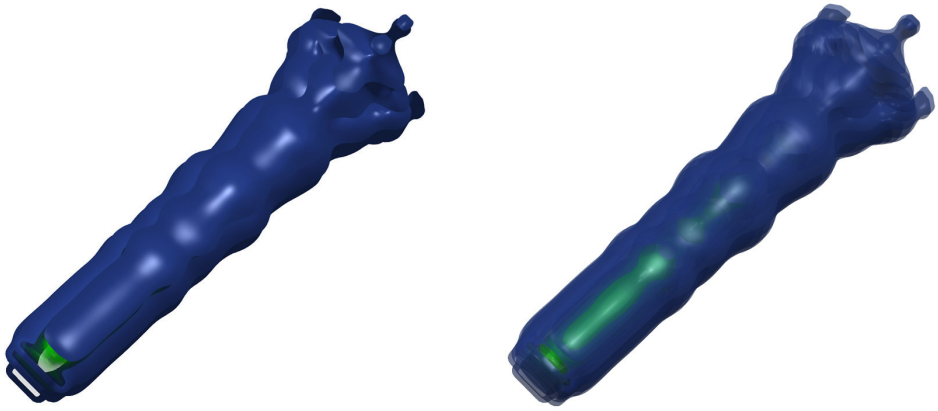


Figure 5.3: *Comparison between α (left) and exponential extinction (right). α-based extinction shows hard transitions and complete opaqueness whereas exponential extinction shows the typical smooth transitions. Dataset courtesy of the German Research Council.*

5.4 Implementation

Figure 5.4 shows the basic pipeline of our implementation. The input volume consists of voxels arranged in a regular grid. Each voxel has a scalar value v and

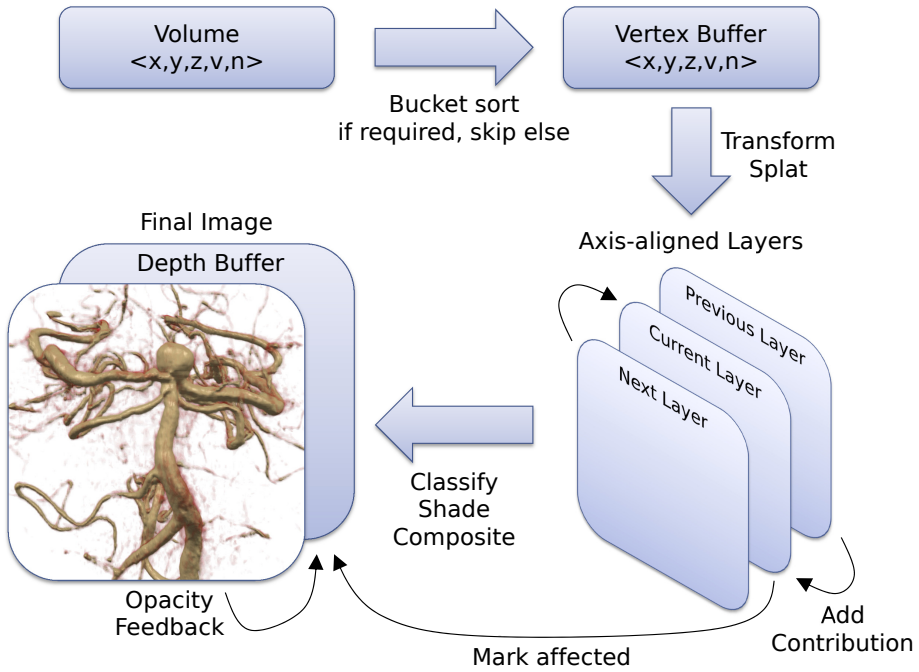


Figure 5.4: *The rendering pipeline of layered volume splatting. The bucket sorting step can be omitted if no* $45°$ *angle is crossed.*

normal n as attributes where the normal is approximated by the gradient computed in a preprocessing step. For any given transfer function, voxels that are classified as transparent can be removed from the dataset resulting in a reduced effective (visible) voxel count.

The first time or when the layer orientation changes, the voxels have to be (re-) distributed to the different layers. A bucket sorting algorithm with run time $O(n)$ is used for that because the voxels only have to be sorted partially based on their layer classification. The output of the bucket sorting is directly written to a vertex buffer object (VBO) in GPU memory. To reduce graphics memory consumption, point sprites are used to represent splats, minimizing the number of vertices to one per voxel. The scalar value v and normal $n = \nabla v/|\nabla v|$ are encoded into the primary color using RGB channels for the normal and the α channel for the scalar value. Besides occasional bucket sorting, which could be pre-cached, rendering does not require any further processing on the CPU. Future frames can be rendered entirely on the GPU unless the view direction exceeds a $45°$ angle provoking an orientation change of the layers.

Splatting is done by iteratively drawing the range of the vertex buffer object for the current layer. The size of the point primitive (footprint extent) is computed

within a vertex shader. This is a difference to view-aligned sheet splatting where every footprint has the same extent inside a sheet.

In order to omit superfluous rasterization of splats and compositing fragments, the same optimizations are used as proposed in [Neophytou and Mueller, 2005]. For the purpose of skipping occluded regions, the order of the rendering is strictly front-to-back. Fragments that become opaque are marked during a separate pass after compositing. The depth-test and the depth-bounds extensions are set in a way that marked pixels are skipped during splatting, reducing rasterization time. Contrariwise, a specific z_i-value is assigned to every layer L_i and written to a shared depth buffer for every rasterized fragment. It enables the compositing pass to only process fragments that have been generated for the current layer L_i.

Adding the contribution of the two adjacent layers to the current one necessitates holding of three consecutive layers $L_{i-1,i,i+1}$ at any time. Splatting therefore works with an intermediary array of three frame buffer objects (FBO). This array of FBOs always contains the last, the current and the next layers in a round robin way. Consequently compositing can be done for layer L_i after splatting layer L_{i+1}.

The final image is composited using a fragment shader program. This shader has manifold tasks. First, the contributions from the two adjacent layers are added according to the pre-integration ratio κ provided to the shader. Once this is completed, the generated fragments are classified and shaded. Finally, the blending is computed using frame buffer readback and the aforementioned exponential extinction function. One can imagine that this fragment shader may be quite expensive. However, large empty areas located completely outside of the volume are cut away using a simple bounding box adapted on the CPU. Furthermore, early-z culling and the depth-bounds extension make sure that the fragment shader program is only executed for the affected fragments. Unfortunately, marking of opaque fragments to exclude them from further processing cannot be done within the compositing shader. Hence another optimization pass is introduced which either marks opaque pixels in the depth buffer or discards the fragments.

5.5 Results and Discussion

For each of the datasets in Table 5.1 a full rotation over 125 frames around a defined axis with a constant angular step is measured. The average frame rate of this rotation is listed in Table 5.2. The rendering is done in a viewport of 512^2 pixels size. All measurements were made on a MacPro with 2x Intel Xeon 2.66GHz processors, 2 GByte of memory, and a GeForce 8800 GT graphics card with 512 MByte of memory. We use the method from [Neophytou and Mueller, 2005] as a benchmark. On equal hardware our implementation of their renderer,

Name	Dimension	Effective splats
Fuel Injection	64^3	14K
Lobster	$301 \times 324 \times 56$	233K
Aneurysm	256^3	79K
Neghip	64^3	122K
Engine	$256^2 \times 128$	1.3M
Skull	256^3	1.4M
Foot	256^3	4.6M
Vertebra	512^3	1.6M

Table 5.1: *Dataset sizes*

Dataset	Axis-Aligned Sheet Splatting	View-Aligned Sheet Splatting	Layered Splatting	3D Texture Slicing
Fuel Injection	39.8fps	47.0fps	100.3fps	197.1fps
Lobster	15.0fps	13.2fps	43.0fps	
Aneurysm	26.7fps	23.0fps	48.7fps	47.3fps
Neghip	7.9fps	11.1fps	30.7fps	192.6fps
Engine	3.8fps	3.0fps	23.2fps	43.0fps
Skull	3.6fps	2.7fps	16.5fps	43.6fps
Foot	0.9fps	0.8fps	8.3fps	40.9fps
Vertebra	3.0fps	2.4fps	14.9fps	

Table 5.2: *Performance results*

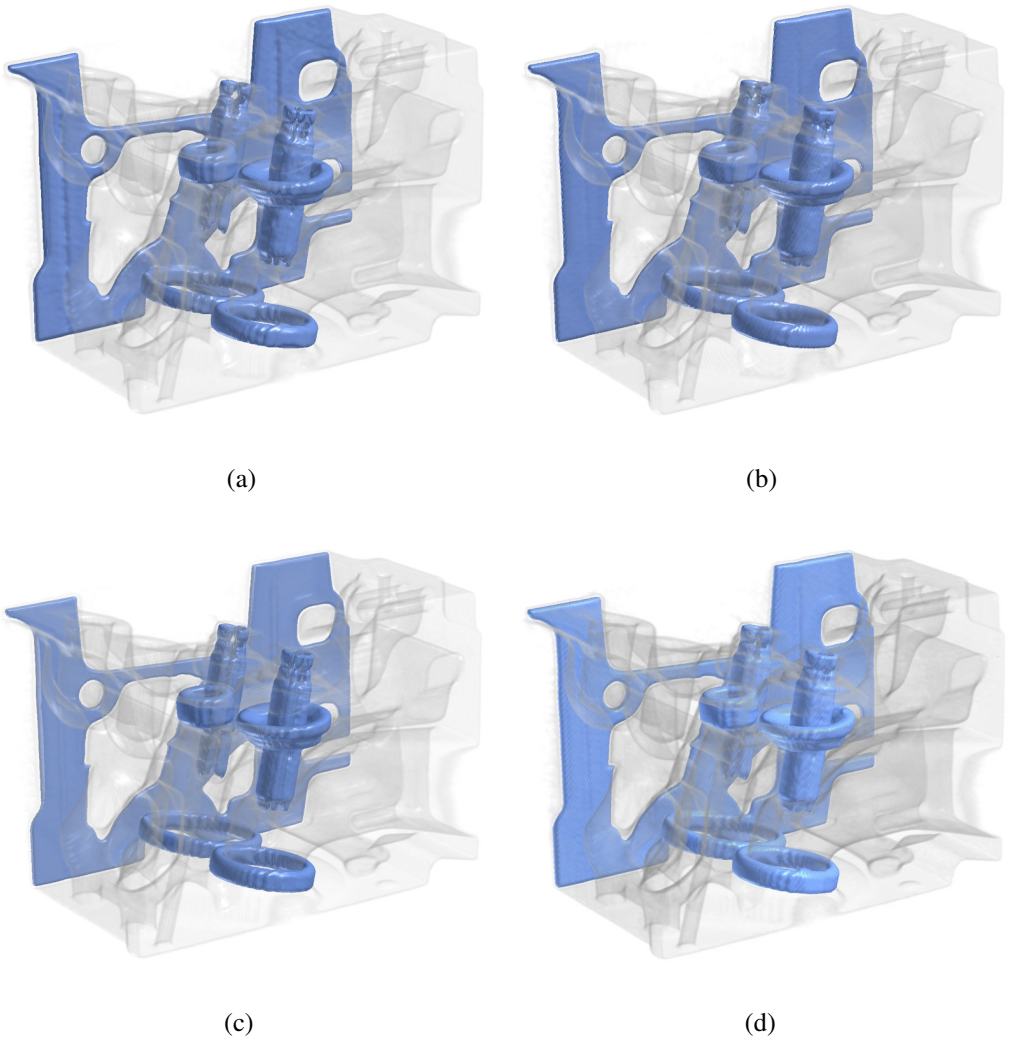
labeled *view-aligned splatter*, achieves the same or slightly higher frame rates.

The experimental results clearly demonstrate the performance of layered splatting, outperforming the other volume splatters, especially when the number of effective splats rises. From a factor 2 compared to the view-aligned sheet splatter for the tiny Fuel Injection dataset, the factor goes up to 10 for the larger multimillion-splat datasets. Generally, the performance of either splatter correlates quite well to the number of effective splats but on different levels. This is also an indicator that on today's hardware the compositing of the sheets or layers is not a limiting factor anymore despite the per-pixel post-classification and shading.

Figure 5.5 shows the engine dataset rendered with a partially transparent transfer function using the renderers from Table 5.2. The image of the layered splatter features a quality comparable to the images from the two other splatters. On a

closeup look the axis-aligned splatter shows a micro-structure on the back plate and the view-aligned splatter and 3D texture slicing show visible artifacts from the sheets. Further, the smoothing effect from the large Gaussian interpolation kernel is obvious with axis-aligned and view-aligned splatting when looking at the valve guides and cannot be matched by layered splatting. Contrariwise layered splatting has the cleanest representation of the thin back plate with no artifacts at all.

Another quality comparison between layered splatting and high-quality view-aligned splatting is presented in Figure 5.6 featuring different medical datasets. To avoid visible artifacts from the sheets when using view-aligned splatting 2x z-super-sampling must be enabled at the cost of a considerable performance impact. Otherwise spurious artifacts from the sheets are visible, especially with the chest dataset. This is not the case with layered splatting unless the dataset has a very small resolution such as 64^3.



(a) (b)

(c) (d)

Figure 5.5: *Quality comparison of the engine dataset with axis-aligned splatting (a), view-aligned splatting (b), layered splatting (c), and 3D texture slicing (d). Dataset courtesy of General Electric.*

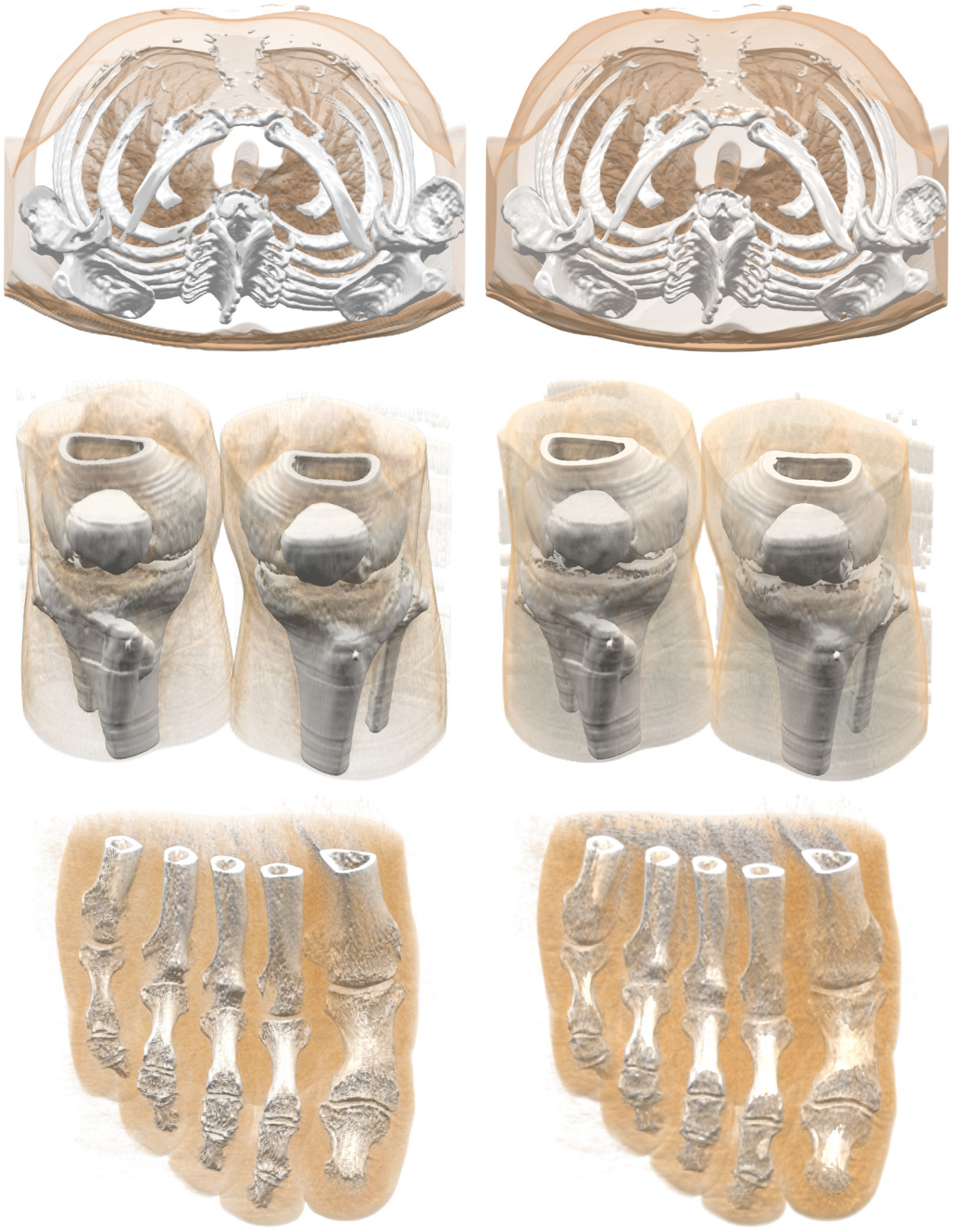


Figure 5.6: *Quality comparison of the chest, knee, and feet dataset. The images in the left column are rendered with layered splatting and in the right column with view-aligned splatting with 2x z-super sampling enabled to avoid visible artifacts from the sheets. Layered splatting with a comparable image quality to view-aligned splatting is approximately 8-10 times faster for these datasets. Datasets courtesy of the Department of Radiology, University of Iowa, USA and Philips Research, Germany respectively.*

CHAPTER 6

CONCLUSION

6.1 Summary

Direct, interactive volume visualization methods have become an important tool in many domains. In this thesis we addressed some of the issues still present with direct volume visualization, and proposed further enhancements making it even more attractive.

The possibility to load a raw dataset obtained from a scanning device or a numerical simulation directly into a volume visualization system without extracting surfaces or other preprocessing steps is one of the biggest advantages of direct volume visualization. At the same time it is one of the biggest issues since there is a priori no way how to deduce a meaningful image from a raw volumetric dataset. It is therefore required to specify a transfer function, which is the explicit mapping from the scalar data of the volumetric dataset to the color and opacity space. However, the manual construction of a transfer function is a difficult and time consuming task. The vast majority of existing automatic solutions focus on specific properties of the volumetric dataset targeting specific application domains. Other methods are semi-automatic involving the user to direct the search towards a single, final transfer function. In contrast we propose visibility-difference entropy transfer function generation, which is an automatic general purpose approach for generating a set of best, distinct transfer functions. This allows for quick insight into each facet of an unknown dataset. Further, our method is designed with regard to highly interactive volume visualization sessions in contrast to the analysis

of static images. The basis for visibility-difference entropy transfer function generation is the visibility-difference entropy metric. It measures the information content of the differential images from different viewpoint images generated with a particular transfer function applied. Including a penalty for overly noisy images and images covering only small parts of the dataset, the metric is an indicator of how much from the dataset is revealed when animated in an interactive session. A basis transfer function has to be selected in the beginning, and on top of that basis function the best distinct transfer functions are generated in a feedback loop based on the visibility-difference entropy metric and a combination of gradient ascent and simulated annealing optimization. Finally, the set of transfer functions is presented to the user providing him a complete overview of the dataset.

Once a satisfying transfer function has been selected, the image needs to be synthesized with an appropriate method. In doing so the illumination and shading model applied plays an important role. Using a simple model carries the risks of small details getting lost in uniformly colored areas and insufficient shading leading to a poor depth perception. Sophisticated, physically accurate models on the other hand are very expensive to compute and not suitable for interactive applications. In this thesis we introduced extinction-based shading and illumination, which is a model incorporating effects such as soft shadows, scattering, ambient occlusion, and color bleeding. Even though these effects are not physically accurate, they are visually plausible with the advantage of being fast enough for interactive applications. With this model, the images in an interactive volume visualization session look more vivid, realistic, and provide a better depth perception helping to better spot small details. Unlike existing methods, our method has no limitations when it comes to the number, placement and type of the light sources within a scenario. The observation is, that basically all effects stem from the extinction of light traveling through the volumetric dataset. By using the original, exponential extinction coefficient of the volume rendering integral instead of the common simplification leading to α-blending, the effects can be integrated into a unified model. Since the original, exponential extinction coefficient is additive and order-independent, a summed area table serves as a basis for the unified model. Soft shadows and scattering effects can then be approximated by a series of queries from the SAT, filling extinction cones towards the light source. These extinction cones estimate the extinction of the light in the neighborhood of a ray from a light source to a voxel. Ambient occlusion and color bleeding can be approximated by a series of shells from the SAT for estimating the extinction of the light in the neighborhood of a voxel. As a result soft shadows including scattering, ambient occlusion, and color bleeding can be computed very quickly greatly enhancing the overall perception of the images.

Volume visualization methods based on 3D textures such as GPU volume raycasting or 3D texture slicing rely on the support for 3D textures of the graphics

hardware. These 3D textures inherently use trilinear interpolation as texture filtering method, which is very simple and fast but not the best choice in terms of quality. However, using different interpolation filters is easily possible with volume splatting. The rise of programmable GPUs boosted the performance of volume splatting as well as the flexibility of implementing different interpolation kernels. However, the major bottleneck in terms of performance remains the rasterization bound of the graphics hardware. The amount of rasterization operations is directly related to the size of the interpolation kernel and therefore bigger interpolation kernels even exacerbate the problem. For high quality splatting in particular, interpolation kernels have to be cut into a number of slabs and each slab has to be splatted onto the appropriate slice. In this thesis we introduced layered splatting, which is able to greatly enhance to performance of volume splatting by deferring the rasterization bound. The basic idea is to not explicitly splat all slabs of an interpolation kernel anymore. In conjunction with a special, cubic interpolation kernel only the main slab is splatted onto the respective slice and the contributions of the other slabs to adjacent slices are approximated using a correction term (virtual splatting). We call therefore the main slice a layer. In addition, our method is able to enhance the quality of splatting by using the original, exponential extinction coefficient of the volume rendering integral instead of the α-blending simplification. Layered splatting is able to achieve a similar image quality compared to state of the art post-classified view-aligned volume splatting while being an order of magnitude faster.

6.2 Future Work

In this thesis some problems of direct volume visualization were addressed and enhancements were proposed. Despite this progress, there are still many challenges left and many more exciting enhancements wait to be discovered. Some of the issues and possible enhancements are presented in the following as directions for further work:

- **Automatic Transfer Function Generation with Harmonic Colors**

  In this thesis we presented a general purpose approach for generating a set of best transfer functions. So far the focus was on the opacity channel of the transfer function because it is the channel specifying what parts are visible, what parts are transparent and what parts are hidden. As the examples of the pre-segmented datasets show, color can be important as well. In future work visibility-difference entropy transfer function generation could be extended to include color. However, the challenges are manifold. Handling of RGB values instead of gray values results in an extended alphabet of 2^{24} instead of

2^8 entries. Further, harmonic color schemas have to be defined as presented in previous work [Wang and Mueller, 2008; Zhou and Takatsuka, 2009] but based on the basis transfer functions. Most important, though, without restrictions the parametric space will be four times as big (four instead of one channel) making an efficient search impossible.

A possible way to go is to predefine a relatively large set of harmonic color transfer functions excluding the opacity and independent of the basis transfer function. This set can then be used as a single additional parameter in the parametric space of the basis transfer function maintaining an efficient search. Additionally, instead of using the entire RGB space as alphabet, the colors can be represented by luminance and chrominance components where the chrominance component may be subsampled.

- **Light Source Shapes**

  As volume visualization methods become mainstream and are not purely used for scientific visualization anymore but move into the entertainment domain as well, illumination effects become even more important. Extinction-based shading and illumination as presented in this thesis can only handle the standard light source types such as point, spot, and area light sources. Interesting effects could be achieved with light sources exhibiting exotic shapes such as a spiraled neon tube.

  The challenge is to incorporate such light sources into extinction-based shading and illumination without affecting the performance too much. A solution could be not to use cones when evaluating the extinction towards such a light source but a shape derived from the projection of the light source in direction of the particular voxel. This specific shape then needs to be approximated appropriately by a number of cuboids. Subsequently the cuboids can be evaluated similar to standard extinction-based shading and illumination.

- **Diffuse Reflections and Refractions**

  In offline ray-tracers for triangle meshes or volumetric datasets, (multiple-) reflections and refractions are standard. In the simplest case these reflections and refractions are computed by recursively tracing reflection or refraction rays. However, recursively tracing rays is not an option for interactive applications due to the computational cost. Other ways need to be discovered to implement these effects into interactive volume visualization systems. Possibly the idea of extinction-based shading and illumination could be used to achieve diffuse first order reflections and refractions (i.e.

when looking through a block of ice). Surfaces in question could be identified by the first and second order derivatives and the material properties for particular voxels. Then, cones approximated by cuboids from the color SAT originally used for color bleeding could be evaluated in direction of the reflection or refraction. The result could look like diffuse reflections or refractions as observable on a block of ice.

- **Handling of Large Datasets**

  A topic becoming more and more important in volume visualization is the handling of very large datasets. Scanning devices have made tremendous progress in recent years producing larger and larger datasets. The problem is that if a scanning device doubles the resolution per axis, the resulting dataset is eight times as large (2^3). This growth rate cannot be matched by the growth rate of the memory on the graphics hardware. Even though many approaches have been presented to solve the problem such as compression methods, bricking approaches, and level-of-detail, each has certain drawbacks. Either the quality suffers (lossy compression such as S3 texture compression [Shreiner et al., 2007]) or the performance suffers (bricking) or an expensive preprocessing is required (certain LOD methods) objecting the desire for immediate, highly interactive volume visualization. Therefore the handling of large datasets is an omnipresent topic for future work.

- **Reference for Volume Visualization Systems**

  A problem in volume visualization is that there is no such thing as a system that could be used as a reference for future quality improvements. In contrast to performance improvements that can be measured in terms of time on equal hardware under equal conditions, quality improvements are mainly evaluated subjectively by researchers through investigation of resulting images. Future work should go in the direction of developing a reference system including a deterministic regression set. Each quality improvement should first pass the entire regression set and take it as a reference before further consideration through researchers. The reference system can then be extended if the majority of researchers agree on the importance of an improvement.

While working on this thesis direct volume visualization has proven to be a very useful and many times a very impressive tool for analyzing volumetric datasets. A large part is attributed to the good quality and especially the high degree of interactivity of today's solutions on today's hardware. We believe that with the ongoing progress of the graphics hardware and algorithms direct volume

visualization will become even more important and also find its way into many domains outside of the scientific visualization community.

BIBLIOGRAPHY

[Appel, 1968] Appel, A. (1968). Some techniques for shading machine renderings of solids. In *Proceedings of the Spring Joint Computer Conference*, pages 37–45.

[Arens and Domik, 2010] Arens, S. and Domik, G. (2010). A survey of transfer functions suitable for volume rendering. In *Proceedings Eurographics/IEEE VGTC Symposium on Volume Graphics*, pages 77–83.

[Bajaj et al., 1997] Bajaj, C. L., Pascucci, V., and Schikore, D. R. (1997). The contour spectrum. In *Proceedings of the Conference on Visualization*, pages 167–175.

[Behrens and Ratering, 1998] Behrens, U. and Ratering, R. (1998). Adding shadows to a texture-based volume renderer. In *Proceedings IEEE Symposium on Volume Visualization*, pages 39–46.

[Blinn, 1977] Blinn, J. F. (1977). Models of light reflection for computer synthesized pictures. *SIGGRAPH Computer Graphics*, 11(2):192–198.

[Bruckner and Gröller, 2007] Bruckner, S. and Gröller, E. (2007). Enhancing depth-perception with flexible volumetric halos. *IEEE Transactions on Visualization and Computer Graphics*, 13:1344–1351.

[Caban and Rheingans, 2008] Caban, J. J. and Rheingans, P. (2008). Texture-based transfer functions for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1364–1371.

[Cabral et al., 1994] Cabral, B., Cam, N., and Foran, J. (1994). Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings of the Symposium on Volume Visualization*, pages 91–98.

[Cameron and Undrill, 1992] Cameron, G. G. and Undrill, P. E. (1992). Rendering volumetric medical image data on a simd-architecture computer. In *Proceedings of the Third Eurographics Workshop on Rendering*, pages 135–145.

[Carlbom, 1993] Carlbom, I. (1993). Optimal filter design for volume reconstruction and visualization. In *Proceedings of the Conference on Visualization*, pages 54–61.

[Chen et al., 2004] Chen, W., Ren, L., Zwicker, M., and Pfister, H. (2004). Hardware-accelerated adaptive EWA volume splatting. In *Proceedings of IEEE Visualization*, pages 67–74.

[Cohen and Greenberg, 1985] Cohen, M. F. and Greenberg, D. P. (1985). The hemi-cube: a radiosity solution for complex environments. *SIGGRAPH Computer Graphics*, 19(3):31–40.

[Correa and Ma, 2008] Correa, C. D. and Ma, K.-L. (2008). Size-based transfer functions: A new volume exploration technique. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1380–1387.

[Correa and Ma, 2009] Correa, C. D. and Ma, K.-L. (2009). Visibility-driven transfer functions. In *Proceedings of the IEEE Pacific Visualization Symposium*, pages 177–184.

[Correa and Ma, 2011] Correa, C. D. and Ma, K.-L. (2011). Visibility histograms and visibility-driven transfer functions. *IEEE Transactions on Visualization and Computer Graphics*, 17(2):192–204.

[Crawfis and Max, 1993] Crawfis, R. and Max, N. (1993). Texture splats for 3D scalar and vector field visualization. In *Proceedings IEEE Visualization*, pages 261–266.

[Crow, 1984] Crow, F. C. (1984). Summed-area tables for texture mapping. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH, pages 207–212.

[Díaz et al., 2010] Díaz, J., Vázquez, P.-P., Navazo, I., and Duguet, F. (2010). Real-time ambient occlusion and halos with summed area tables. *Computers & Graphics*, 34(4):337–350.

[Dubois and Rodrigue, 1977] Dubois, P. and Rodrigue, G. (1977). An analysis of the recursive doubling algorithm. In *High Speed Computer and Algorithm Organization*, pages 299–305.

[Duch and Jankowski, 1997] Duch, W. and Jankowski, N. (1997). New neural transfer functions. *Neural Computing Surveys*, 7:639–658.

[Dutre et al., 2002] Dutre, P., Bala, K., and Bekaert, P. (2002). *Advanced Global Illumination*. AK Peters.

[Dyken et al., 2007] Dyken, C., Ziegler, G., Theobalt, C., and Seidel, H.-P. (2007). GPU marching cubes on shader model 3.0 and 4.0. Research Report MPI-I-2007-4-006, Max-Planck-Institut für Informatik.

[Engel et al., 2006] Engel, K., Hadwiger, M., Kniss, J. M., Rezk-Salama, C., and Weiskopf, D. (2006). *Real-Time Volume Graphics*. AK Peters.

[Engel et al., 2001] Engel, K., Kraus, M., and Ertl, T. (2001). High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, pages 9–16.

[Everitt, 2001] Everitt, C. (2001). Interactive order-independent transparency. Technical report, NVIDIA Corp.

[Fang et al., 1998] Fang, S., Tom, B., and Tuceryan, M. (1998). Image-based transfer function design for data exploration in volume visualization. In *Proceedings of the Conference on Visualization*, pages 319–326.

[Feixas et al., 2009] Feixas, M., Sbert, M., and González, F. (2009). A unified information-theoretic framework for viewpoint selection and mesh saliency. *ACM Transactions on Applied Perception*, 6(1):1–23.

[Foley et al., 1990] Foley, J. D., van Dam, A., Feiner, S. K., and Hughes, J. F. (1990). *Computer graphics: principles and practice*. Addison-Wesley Longman Publishing Co., Inc.

[Fourier, 1822] Fourier, J. B. J. (1822). *Théorie analytique de la chaleur*. Didot.

[Fujishiro et al., 1999] Fujishiro, I., Azuma, T., and Takeshima, Y. (1999). Automating transfer function design for comprehensible volume rendering based on 3D field topology analysis. In *Proceedings of the Conference on Visualization*, pages 467–470.

[Gonzalez and Woods, 2006] Gonzalez, R. C. and Woods, R. E. (2006). *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc.

[Goral et al., 1984] Goral, C. M., Torrance, K. E., Greenberg, D. P., and Battaile, B. (1984). Modeling the interaction of light between diffuse surfaces. *SIGGRAPH Computer Graphics*, 18:213–222.

[Grau and Tost, 2007] Grau, S. and Tost, D. (2007). Image-space sheet-buffered splatting on the GPU. In *International Conference on Computer Graphics and Visualization*, pages 75–82.

[Hadwiger et al., 2009] Hadwiger, M., Ljung, P., Rezk-Salama, C., and Ropinski, T. (2009). Advanced illumination techniques for GPU-based volume raycasting. ACM SIGGRAPH Course Notes.

[Hall et al., 1992] Hall, L., Bensaid, A., Clarke, L., Velthuizen, R., Silbiger, M., and Bezdek, J. (1992). A comparison of neural network and fuzzy clustering techniques in segmenting magnetic resonance images of the brain. *IEEE Transactions on Neural Networks*, 3(5):672–682.

[He et al., 1996] He, T., Hong, L., Kaufman, A., and Pfister, H. (1996). Generation of transfer functions with stochastic search techniques. In *Proceedings of the Conference on Visualization*, pages 227–234.

[Heineman et al., 2008] Heineman, G., Pollice, G., and Selkow, S. (2008). *Algorithms in a Nutshell*. O'Reilly Media, Inc.

[Hensley et al., 2005] Hensley, J., Scheuermann, T., Coombe, G., Singh, M., and Lastra, A. (2005). Fast summed-area table generation and its applications. *Computer Graphics Forum*, 24:547–555.

[Henyey and Greenstein, 1941] Henyey, L. G. and Greenstein, J. L. (1941). Diffuse radiation in the galaxy. *Astrophysical Journal*, 93:70–83.

[Hernell et al., 2010] Hernell, F., Ljung, P., and Ynnerman, A. (2010). Local ambient occlusion in direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 16(4):548–559.

[Hsu and Marzetta, 1989] Hsu, K. and Marzetta, T. L. (1989). Velocity filtering of acoustic well logging waveforms. In *IEEE Transactions on Acoustics, Speech and Signal Processing*, volume 37, pages 265–274.

[Huang et al., 2000] Huang, J., Crawfis, R., Shareef, N., and Mueller, K. (2000). Fastsplats: optimized splatting on rectilinear grids. In *Proceedings IEEE Visualization*, pages 219–226.

[Itti et al., 1998] Itti, L., Koch, C., and Niebur, E. (1998). A model of saliencey-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259.

[Jänicke and Chen, 2010] Jänicke, H. and Chen, M. (2010). A salience-based quality metric for visualization. *Computer Graphics Forum*, 29(3):1183–1192.

[Jensen, 2001] Jensen, H. W. (2001). *Realistic image synthesis using photon mapping*. AK Peters.

[Jensen and Christensen, 2007] Jensen, H. W. and Christensen, P. (2007). High quality rendering using ray tracing and photon mapping. In *ACM SIGGRAPH Courses*.

[Kajiya and von Herzen, 1984] Kajiya, J. T. and von Herzen, B. P. (1984). Ray tracing volume densities. In *Proceedings ACM SIGGRAPH*, pages 165–174.

[Kindlmann and Durkin, 1998] Kindlmann, G. and Durkin, J. W. (1998). Semi-automatic generation of transfer functions for direct volume rendering. In *Proceedings of the IEEE Symposium on Volume Visualization*, pages 79–86.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

[Kniss et al., 2001] Kniss, J., Kindlmann, G., and Hansen, C. (2001). Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings of the Conference on Visualization*, pages 255–262.

[Kniss et al., 2002a] Kniss, J., Kindlmann, G., and Hansen, C. (2002a). Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285.

[Kniss et al., 2002b] Kniss, J., Premoze, S., Hansen, C., and Ebert, D. (2002b). Interactive translucent volume rendering and procedural modeling. In *Proceedings IEEE Visualization*, pages 109–116.

[König and Gröller, 2001] König, A. and Gröller, M. E. (2001). Mastering transfer function specification by using volumepro technology. In *Proceedings of the Spring Conference on Computer Graphics*, pages 279–286.

[Kruger and Westermann, 2003] Kruger, J. and Westermann, R. (2003). Acceleration techniques for GPU-based volume rendering. In *Proceedings IEEE Visualization*, pages 287–292.

[La Mar et al., 1999] La Mar, E. C., Hamann, B., and Joy, K. I. (1999). Multiresolution techniques for interactive texture-based volume visualization. In *Proceedings of the IEEE Visualization Conference*, pages 355–361.

[Lacroute and Levoy, 1994] Lacroute, P. and Levoy, M. (1994). Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques*, pages 451–458.

[Laga, 2010] Laga, H. (2010). Semantics-driven approach for automatic selection of best views of 3D shapes. In *Proceedings Eurographics Workshop on 3D Object Retrieval*, pages 15–22.

[Landis, 2002] Landis, H. (2002). Production-ready global illumination. In *SIGGRAPH Course Notes*, volume 16.

[Levoy, 1988] Levoy, M. (1988). Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37.

[Li et al., 2009] Li, K., Qi, R., Xiao, D., Yang, L., and Li, Z. (2009). Applying particle swarm optimization to transfer function specification for direct volume rendering. In *Proceedings of the International Joint Conference on Bioinformatics, Systems Biology and Intelligent Computing*, pages 573–576.

[Lindholm et al., 2001] Lindholm, E., Kilgard, M. J., and Moreton, H. (2001). A user-programmable vertex engine. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH, pages 149–158.

[Ljung, 2006] Ljung, P. (2006). Adaptive sampling in single pass, GPU-based raycasting of multiresolution volumes. In *Proceedings Eurographics/IEEE International Workshop on Volume Graphics*, pages 39–46.

[Lorensen and Cline, 1987] Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Computer Graphics*, 21(4):163–169.

[Lundstrom et al., 2006] Lundstrom, C., Ljung, P., and Ynnerman, A. (2006). Local histograms for design of transfer functions in direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 12:1570–1579.

[Marks et al., 1997] Marks, J., Andalman, B., Beardsley, P. A., Freeman, W., Gibson, S., Hodgins, J., Kang, T., Mirtich, B., Pfister, H., Ruml, W., Ryall, K., Seims, J., and Shieber, S. (1997). Design galleries: a general approach to setting parameters for computer graphics and animation. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques*, pages 389–400.

[Marschner and Lobb, 1994] Marschner, S. R. and Lobb, R. J. (1994). An evaluation of reconstruction filters for volume rendering. In *Proceedings of the Conference on Visualization*, pages 100–107.

[Max, 1995] Max, N. (1995). Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108.

[McGillem and Cooper, 1984] McGillem, C. D. and Cooper, G. R. (1984). *Continuous and Discrete Signal and System Analysis*. Holt, Reinhart and Winston.

[Méndez et al., 2003] Méndez, A., Sbert, M., and Catà, J. (2003). Real-time obscurances with color bleeding. In *Proceedings Spring Conference on Computer Graphics*, pages 171–176.

[Méndez-Feliu and Sbert, 2008] Méndez-Feliu, À. and Sbert, M. (2008). From obscurances to ambient occlusion: A survey. *The Visual Computer*, 25(2):181–196.

[Mertz and Gray, 1934] Mertz, P. and Gray, F. (1934). A theory of scanning and its relation to the characteristics of the transmitted signal in telephotography and television. *Bell System Technical Journal*, 13:464–514.

[Mitchell and Netravali, 1988] Mitchell, D. P. and Netravali, A. N. (1988). Reconstruction filters in computer-graphics. *SIGGRAPH Computer Graphics*, 22(4):221–228.

[Moreland, 2004] Moreland, K. D. (2004). *Fast High Accuracy Volume Rendering*. PhD thesis, The University of New Mexico.

[Mortara and Spagnuoloa, 2009] Mortara, M. and Spagnuoloa, M. (2009). Semantics-driven best view of 3D shapes. *Computers & Graphics*, 33(3):280–290.

[Mueller and Crawfis, 1998] Mueller, K. and Crawfis, R. (1998). Eliminating popping artifacts in sheet buffer-based splatting. In *Proceedings IEEE Visualization*, pages 239–246.

[Mueller et al., 1999] Mueller, K., Möller, T., and Crawfis, R. (1999). Splatting without the blur. In *Proceedings IEEE Visualization*, pages 363–370.

[Nakagawa et al., 2006] Nakagawa, M., Takata, M., and Joe, K. (2006). Automatic viewpoint selection for a visualization I/F in a PSE. In *Proceedings IEEE International Conference on e-Science and and Grid Computing*, pages 100–106.

[Neophytou and Mueller, 2005] Neophytou, N. and Mueller, K. (2005). GPU accelerated image aligned splatting. In *Eurographics/IEEE VGTC Workshop on Volume Graphics*, pages 197–205.

[Neophytou et al., 2006] Neophytou, N., Mueller, K., McDonnell, K. T., Hong, W., Guan, X., Qin, H., and Kaufman, A. E. (2006). GPU-accelerated volume splatting with elliptical RBFs. In *IEEE VGTC Symposium on Visualization*, pages 13–20.

[NVIDIA, 2006] NVIDIA (2006). *GeForce 8800 GPU Architecture Overview*.

[Nyquist, 1928] Nyquist, H. (1928). Certain topics in telegraph transmission theory. *Transactions of the AIEE*, 47:617–644.

[Phong, 1973] Phong, B. T. (1973). *Illumination for computer-generated images*. PhD thesis, The University of Utah.

[Porter and Duff, 1984] Porter, T. and Duff, T. (1984). Compositing digital images. In *Proceedings ACM SIGGRAPH*, pages 253–259.

[Rezk Salama et al., 2006] Rezk Salama, C., Keller, M., and Kohlmann, P. (2006). High-level user interfaces for transfer function design with semantics. *IEEE Transactions on Visualization and Computer Graphics*, 12:1021–1028.

[Roettger et al., 2003] Roettger, S., Guthe, S., Weiskopf, D., Ertl, T., and Strasser, W. (2003). Smart hardware-accelerated volume rendering. In *Proceedings of the Symposium on Data Visualisation*, pages 231–238.

[Ropinski et al., 2010] Ropinski, T., Döring, C., and Rezk-Salama, C. (2010). Interactive volumetric lighting simulating scattering and shadowing. In *Proceedings IEEE Pacific Visualization Symposium*, pages 169–176.

[Ropinski et al., 2008a] Ropinski, T., Kasten, J., and Hinrichs, K. (2008a). Efficient shadows for GPU-based volume raycasting. In *Proceedings International Conference on Computer Graphics, Visualization and Computer Vision (WSCG)*, pages 17–24.

[Ropinski et al., 2008b] Ropinski, T., Meyer-Spradow, J., Diepenbrock, S., Mensmann, J., and Hinrichs, K. H. (2008b). Interactive volume rendering with dynamic ambient occlusion and color bleeding. *Computer Graphics Forum*, 27(2):567–576.

[Rost, 2006] Rost, R. J. (2006). *OpenGL Shading Language, Second Edition*. Addison-Wesley Professional.

[Ruiz et al., 2008] Ruiz, M., Boada, I., Viola, I., Bruckner, S., Feixas, M., and Sbert, M. (2008). Obscurance-based volume rendering framework. In *Proceedings IEEE/EG Symposium on Volume and Point-Based Graphics*, pages 113–120.

[Samet, 1990] Samet, H. (1990). *The design and analysis of spatial data structures*. Addison-Wesley Longman Publishing Co., Inc.

[Sato et al., 2000] Sato, Y., Westin, C.-F., Bhalerao, A., Nakajima, S., Shiraga, N., Tamura, S., and Kikinis, R. (2000). Tissue classification based on 3D local intensity structures for volume rendering. In *IEEE Transactions on Visualization and Computer Graphics*, pages 160–180.

[Scharsach, 2005] Scharsach, H. (2005). Advanced GPU raycasting. In *Proceedings Central European Seminar on Computer Graphics (CESCG)*, pages 69–76.

[Schott et al., 2009] Schott, M., Pegoraro, V., Hansen, C., Boulanger, K., and Bouatouch, K. (2009). A directional occlusion shading model for interactive direct volume rendering. *Computer Graphics Forum*, 28(3):855–862.

[Shanmugam and Arikan, 2007] Shanmugam, P. and Arikan, O. (2007). Hardware accelerated ambient occlusion techniques on GPUs. In *Proceedings Symposium on Interactive 3D Graphics and Games*, pages 73–80. ACM SIGGRAPH.

[Shannon, 1948] Shannon, C. E. (1948). A mathematical theory of communication. In *The Bell System Technical Journal*, volume 27, pages 379–423, 623–656.

[Shannon, 1949] Shannon, C. E. (1949). Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21.

[Shreiner et al., 2007] Shreiner, D., Woo, M., Neider, J., and Davis, T. (2007). *OpenGL Programming Guide, Sixth Edition*. Addison-Wesley Professional.

[Sillion and Puech, 1994] Sillion, F. X. and Puech, C. (1994). *Radiosity and Global Illumination*. Morgan Kaufmann Publishers Inc.

[Snyman, 2005] Snyman, J. (2005). *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms*. Springer.

[Takeshima et al., 2005] Takeshima, Y., Takahashi, S., Fujishiro, I., and Nielson, G. (2005). Introducing topological attributes for objective-based visualization of simulated datasets. *International Workshop on Volume Graphics*, 0:137–236.

[Tao et al., 2009] Tao, Y., Lin, H., Bao, H., Dong, F., and Clapworthy, G. (2009). Structure-aware viewpoint selection for volume visualization. In *Proceedings IEEE Pacific Visualization Symposium*, pages 193–200.

[Theussl et al., 2000] Theussl, T., Hauser, H., and Gröller, E. (2000). Mastering windows: improving reconstruction. In *Proceedings of the IEEE Symposium on Volume Visualization*, pages 101–108.

[Tuy and Tuy, 1984] Tuy, H. and Tuy, L. (1984). Direct 2D display of 3D objects. *IEEE Computer Graphics and Applications*, 4(10):29–34.

[Tzeng et al., 2003] Tzeng, F.-Y., Lum, E. B., and Ma, K.-L. (2003). A novel interface for higher-dimensional classification of volume data. In *Proceedings of the IEEE Visualization*, pages 505–512.

[Udupa and Herman, 1999] Udupa, J. K. and Herman, G. T. (1999). *3D Imaging in Medicine (2nd edition)*. CRC Press.

[Vázquez et al., 2004] Vázquez, P.-P., Feixas, M., Sbert, M., and Heidrich, W. (2004). Automatic view selection using viewpoint entropy and its application to image-based modelling. *Computer Graphics Forum*, 22(4):689–700.

[Veach and Guibas, 1997] Veach, E. and Guibas, L. J. (1997). Metropolis light transport. In *SIGGRAPH Computer Graphics*, pages 65–76.

[Šereda et al., 2006] Šereda, P., Vilanova Bartroli, A., Serlie, I. W. O., and Gerritsen, F. A. (2006). Visualization of boundaries in volumetric data sets using lh histograms. In *IEEE Transactions on Visualization and Computer Graphics*, volume 12, pages 208–218.

[Wallace et al., 1987] Wallace, J. R., Cohen, M. F., and Greenberg, D. P. (1987). A two-pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques*, pages 311–320.

[Wallis et al., 1989] Wallis, J., Miller, T., Lerner, C., and Kleerup, E. (1989). Three-dimensional display in nuclear medicine. *IEEE Transactions on Medical Imaging*, 8(4):297–303.

[Wang and Mueller, 2008] Wang, L. and Mueller, K. (2008). Harmonic colormaps for volume visualization. In *IEEE/EG Symposium on Volume Graphics*, pages 30–40.

[Weber and Scheuermann, 2004] Weber, G. H. and Scheuermann, G. (2004). *Geometric Modeling for Scientific Visualization*, chapter Automating Transfer Function Design Based on Topology Analysis, pages 293–306. Springer.

[Westover, 1989] Westover, L. (1989). Interactive volume rendering. In *Proceedings Workshop on Volume Visualization*, pages 9–16.

[Westover, 1990] Westover, L. (1990). Footprint evaluation for volume rendering. In *Proceedings ACM SIGGRAPH*, pages 367–376.

[Whitted, 1979] Whitted, T. (1979). An improved illumination model for shaded display. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques*.

[Williams, 1978] Williams, L. (1978). Casting curved shadows on curved surfaces. In *Proceedings ACM SIGGRAPH*, pages 270–274.

[Wilson et al., 1994] Wilson, O., Van Gelder, A., and Wilhelms, J. (1994). Direct volume rendering via 3D textures. Technical report, University of California at Santa Cruz, CA, USA.

[Wright, 1972] Wright, W. V. (1972). *An interactive computer graphic system for molecular studies*. PhD thesis, The University of North Carolina at Chapel Hill.

[Zhang and Crawfis, 2003] Zhang, C. and Crawfis, R. (2003). Shadows and soft shadows with participating media using splatting. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):139–149.

[Zhang and Sun, 2003] Zhang, J.-W. and Sun, J.-Z. (2003). Adaptive transfer function design for volume rendering by using a general regression neural network. In *International Conference on Machine Learning and Cybernetics*, volume 4, pages 2234–2239.

[Zhou and Takatsuka, 2009] Zhou, J. and Takatsuka, M. (2009). Automatic transfer function generation using contour tree controlled residue flow model and color harmonics. *IEEE Transactions on Visualization and Computer Graphics*, 15:1481–1488.

[Zwicker et al., 2001] Zwicker, M., Pfister, H., van Baar, J., and Gross, M. (2001). EWA volume splatting. In *Proceedings of the Conference on Visualization*, pages 29–36.

CURRICULUM VITAE

Personal Information

Name	Philipp Christoph Schlegel
Date of birth	September 11, 1978
Place of birth	Wetzikon ZH, Switzerland

Education

2006 - 2011	Doctor in Informatics in the Visualization and Multimedia Lab, Department of Informatics, University of Zurich, Switzerland Subject of dissertation: ”Automatic Transfer Function Generation and Extinction-Based Approaches in Direct Volume Visualization” Advisor: Prof. Dr. Renato Pajarola, University of Zurich Co-Examiner: Prof. Dr. Ronald Peikert, ETH Zurich
2004	Dipl. Informatik-Ing. ETH, Department of Computer Science, Swiss Federal Institute of Technology (ETH) Zurich, Switzerland Subject of diploma thesis: ”Generischer Octree zur Repräsentation von impliziten Flächen” Advisor: Prof. Dr. Markus Gross, ETH Zurich

1999 - 2004	Student of Computer Science at the Swiss Federal Institute of Technology (ETH) Zurich, Switzerland
1994 - 1999	Kantonsschule Zürcher Oberland in Wetzikon ZH, Switzerland

Professional Experience

since 2011	Software Engineer GetYourGuide AG, Zurich, Switzerland
2006 - 2011	Application Engineer & Designer UBS AG, Zurich, Switzerland
2005 - 2006	Junior Key People (JKP)/Graduate Training Program (GTP) UBS AG, Zurich, Switzerland
2003 - 2004	4 months internship as Application Developer Credit Suisse, Zurich, Switzerland
1999	2 months internship as IT Administrator Phonak AG, Stäfa, Switzerland

Publications

Conference Publications

P. Schlegel, M. Makhinya, and R. Pajarola. ”Extinction-based Shading and Illumination in GPU Volume Ray-Casting”, IEEE Transactions on Visualization and Computer Graphics. 2011. 17(12):1795-1802.

P. Goswami, P. Schlegel, B. Solenthaler and R. Pajarola. ”Interactive SPH Simulation and Rendering on the GPU”. Proceedings ACM SIGGRAPH/Eurographics Symposium on Computer Animation. 2010. pp. 55-64.

P. Schlegel and R. Pajarola. ”Layered Volume Splatting”. Proceedings International Symposium on Visual Computing. 2009. pp. 1-12.

Misc

P. Schlegel and R. Pajarola. ”Layered Volume Splatting”. Posters IEEE Visualization Conference. 2008.

P. Schlegel. ”Generischer Octree zur Repräsentation von impliziten Flächen”.

Diploma thesis, Computer Graphics Laboratory, ETH Zurich, 2004.

O. Biderbost and P. Schlegel. ”NPDA Browser Lernumgebung für nichtdeterministische Kellerautomaten”. Semester thesis, Institute of Theoretical Computer Science, ETH Zurich, 2003.

C. Schroedter and P. Schlegel. ”Lagersimulation”. Semester thesis, BWI Center for Industrial Management, ETH Zurich, 2003.